UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Pinchapp

Autor/es

ALEJANDRO RUIZ-OLALLA MOURE

Director/es

BEATRIZ PÉREZ VALLE

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Pinchapp

Realizado por:

Alejandro Ruiz-Olalla Moure

Tutelado por:

Beatriz Pérez Valle

Logroño, septiembre 2020

RESUMEN

Hoy en día la mayor parte de las relaciones sociales están tendiendo a llevarse a cabo a través del móvil. Informatizar la gestión de las comidas realizadas entre amigos podría considerarse como otro paso a dar en esta progresión. Este Trabajo Fin de Grado aborda el desarrollo de una aplicación móvil que automatice este proceso de una manera más eficiente, centrándose en el caso particular de la realización de comidas y cenas entre amigos por la Calle Laurel.

La aplicación, la cual hemos denominado “Pinchapp” (*Pincho application*), deberá digitalizar toda la información relacionada con cada una de las quedadas que, a partir de este momento, serán llamadas “eventos”. Desde “Pinchapp” se van a integrar las acciones de solicitudes de amistad entre las diferentes personas a participar en dichos eventos, así como la generación y control de los mismos. La aplicación también proporciona un mapa para poder ver la situación de los bares de La Laurel, de forma que los usuarios pueden escoger a dónde ir, sin necesidad de usar otras aplicaciones. La gestión de eventos abarca desde su creación, generando un “bote” común entre los participantes, hasta el pago en los establecimientos. A la hora de generar cada evento, la persona que lo ha creado, o administrador, será la encargada de pagar y controlar dicho evento. El “bote” se crea al inicio de cada evento, y consiste en la suma de las cantidades a pagar por cada usuario que participa en el mismo, aportando cada uno de ellos la misma cuantía. En el caso de que alguien decida bien incorporarse o bien abandonar el grupo, la aplicación realiza el cálculo del importe proporcional que se tendrá que abonar o reembolsar, actualizando de esta manera el “bote”. Durante la duración de estos eventos, tanto los comensales como el administrador de la quedada pueden visualizar el “bote” total.

En resumen, esta aplicación pretende acercar La Laurel y la cultura de los pinchos a todo el mundo.

ABSTRACT

Nowadays most social relationships are being carried out through mobile phones. Computerizing the management of going out for some food with friends could be considered the very next step in this progression. This Final Degree Project addresses the development of a mobile application that automates this process in a more efficient way, focusing on the particular case of having meals with friends on Calle Laurel*.*

*The application, which has been called "Pinchapp" (*Pincho *application), must digitize all the information related to each of the meetings that, from now on, will be called "events". From “Pinchapp” the actions of friend requests between different users to participate in said events, as well as the generation and control of them, will be integrated. The application also provides a map to see the situation of the bars in* La Laurel*, so that users can choose where to go, without using other applications. Event management ranges from its creation, generating a common "chip-in" among participants, to pay at the establishments. When generating each event, the person who created it, or administrator, will be in charge of paying and controlling said event. The "chip-in" is created at the beginning of each event, and consists of the sum of the amounts to be paid (or chipped in) by each user who participates in it, each contributing the same amount. In the event that someone decides to either join or leave the group, the application calculates the proportional amount that they will have to pay or reimburse, thus updating the "chip in". During the duration of these events, both participants and the event administrator can view the total "chip-in".*

In summary, this application aims to bring La Laurel and pinchos *culture to everyone.*

INDICE

1. INTRODUCCIÓN

1. 1. PRÓLOGO

Vivimos en un mundo cada vez más informatizado, desde la gestión del hogar con la domótica, hasta apps que te indican dónde has aparcado el coche.

A principios del curso académico 2019/20, realicé las prácticas en la empresa GFI, la cual tiene en el sector bancario una de sus principales líneas de trabajo. El auge de los pagos online, como el realizado a través del teléfono móvil, permite abonar el importe correspondiente sin tener que introducir la tarjeta de crédito o débito, gracias a la tecnología NFC [17] (*Near Field Communication* o comunicación de campo cercano).

Dentro de este contexto surge este Trabajo Fin de Grado (TFG), a través del cual vamos a intentar dar una solución tecnológica a un problema que todavía no ha sido explotado en su totalidad. La persona asignada por GFI para tutorizar este proyecto fue Javier Virto, en ese momento, jefe de equipo de la oficina de GFI Logroño, quien se convirtió en cliente del mismo al sugerir el desarrollo de la aplicación abordada en este TFG.

Escogí como nombre para la aplicación “Pinchapp”, que se plantea como una aplicación móvil orientada a la gestión de comidas y cenas entre amigos, en principio, realizadas en la Calle Laurel, y en las que se utiliza el concepto de “bote” (cuando varios compañeros deciden aportar una misma cantidad de dinero para gastarla en las sucesivas compras, principalmente, de carácter gastronómico). Además de gestionar el dinero, la aplicación permitiría pagar en los establecimientos, así como ofrecer la geolocalización de bares y restaurantes en la zona en la que se encuentre el usuario. En el caso de que alguien decida, bien incorporarse, o bien abandonar el grupo, la aplicación realizaría el cálculo del importe proporcional que se tendría que abonar o reembolsar, actualizando de esta manera el “bote”.

1. 2. MOTIVACIÓN

Como se ha comentado previamente, al comienzo del curso inicié las prácticas en la empresa GFI, una compañía centrada en consultorías informáticas que, entre otros, desarrolla proyectos en el sector bancario. Tras estar involucrado en trabajos relacionados con la banca en la empresa, me surgió un gran interés por este tipo de servicios. Durante dichas prácticas, el lenguaje utilizado fue *React*[8] (en la sección *1. 4. Tecnologías* se proporciona más información sobre este lenguaje), lo que motivó una disposición a una mayor formación en este lenguaje informático.

Aunque no directamente relacionado con el sector bancario, pero sí en el mismo contexto, el presente TFG nace de una cena con amigos, principalmente, como propuesta de solución a la gestión del “bote”. Más concretamente en la cultura riojana, es muy común “ir de pinchos” poniendo un “bote”, es decir, salir a tomar algo con amigos, de tal manera que cada uno pone, en una cuenta común, un dinero determinado, que se va gastando a medida que se van pagando las rondas. De esta manera, todos los comensales gastan el mismo dinero, aunque tomen diferentes consumiciones, y se ahorra el engorroso momento de hacer cuentas.

A raíz de comprender este acto tan habitual de salir de copas, y por su importancia para la capital riojana, surge el interés de crear una aplicación que traslade este hecho tan cotidiano a un entorno tecnológico. En este proyecto se quieren transmitir estos conceptos, unirlos y plasmarlos en una aplicación para que la experiencia del usuario sea única, útil e intuitiva.

Mediante este proyecto, se va a intentar automatizar el proceso de recogida del “bote”, además de administrar las transacciones, tanto en los locales para pagar, como entre los usuarios, facilitándoles un mapa para encontrar más rápido los bares a los que poder ir, de forma que dicha funcionalidad de pago esté disponible. En este aspecto, la aplicación se centrará en la Calle Laurel de Logroño.

1. 3. ACTORES

Como en todo proyecto, siempre hay un equipo detrás del trabajo realizado: la empresa, el cliente, etc. En este apartado se determinan quiénes han sido los participantes del mismo.

AUTORÍA

El autor de este TFG es Alejandro Ruiz-Olalla Moure.

TUTORES

Desde la Universidad de La Rioja, Beatriz Pérez Valle ha sido la encargada de tutorizar este trabajo.

Desde GFI, inicialmente el tutor fue Javier Virto, quien también fue el cliente inicial. No obstante, una vez comenzado el proyecto, Javier abandonó la empresa. En ese momento, Ignacio Tobía, jefe de equipo de la sección de sanidad de la oficina GFI Logroño, pasó a ser el nuevo tutor oficial, compartiendo este rol, de una forma más informal, con Joan Roca.

CLIENTES

Como se acaba de adelantar, Javier Virto, al proponer la idea de la aplicación, fue el que implantó el germen de lo que posteriormente se convirtió en este TFG. Como cliente inicial, dio las claves y los conceptos básicos necesarios para el desarrollo de “Pinchapp”. No obstante, aunque la idea inicial fuera de Javier y éste marcara unos requisitos iniciales, éstos eran incompletos. Por ello, al abandonar la empresa al comienzo del proyecto, nos replanteamos los requisitos finales de la aplicación en su conjunto. En ese momento el cliente pasó a ser Alejandro Ruiz-Olalla, con el apoyo de la tutora académica, Beatriz Pérez Valle.

1. 4. TECNOLOGÍAS

En este apartado señalamos las tecnologías que se han analizado para ser utilizadas en este proyecto, destacando las que finalmente se han empleado.

- ***JavaScript*** [31] es un lenguaje de programación utilizado esencialmente en la creación de páginas web dinámicas. Este tipo de páginas son aquellas que incorporan efectos, por ejemplo, un texto que se desvanece, imágenes animadas, o acciones que se realizan al pulsar botones.

  Técnicamente hablando, *JavaScript* es un lenguaje de programación interpretado, es decir, no es necesario que los programas se compilen para poder ejecutarlos. De otra forma, los programas que se desarrollan con *JavaScript* se pueden testar de manera directa en un navegador sin que haya necesidad de ningún proceso intermedio.

  Esta tecnología ha sido utilizada en este TFG.

- ***React***[8] (también llamada *ReactJS*) es una librería de *JavaScript* de código abierto. Está diseñada para facilitar la creación de interfaces de usuario en el desarrollo de aplicaciones web. Esto sirve de ayuda a los programadores para construir apps web que usen datos que vayan actualizándose a medida que éstas se van utilizando. La interfaz de usuario es el único ámbito de la aplicación que usa *React*.

  Las aplicaciones *React* son construidas a partir de componentes, es decir, elementos independientes que se pueden reutilizar y que, de igual manera, describen su visualización y comportamiento.

  El lenguaje *React* no se utilizará en este TFG ya que, como veremos a continuación, al ser nuestro objetivo el desarrollo de una aplicación móvil se empleará *React Native*.

- ***React Native*** [9]. Mientras que *React* es un *framework* para aplicaciones utilizando *JavaScript*, *React Native* es a la vez un *framework* para aplicaciones híbridas y una plataforma entera que permite construir en lenguaje nativo. De esta manera no utiliza HTML para renderizar la aplicación, sino que provee componentes alternativos que funcionan de manera similar. Se basa en la librería de *JavaScript* para crear componentes visuales, pero, en vez de ejecutarlos en el navegador, cambia el propósito de los mismos para que funcionen de manera directa sobre las plataformas móviles nativas.

  En vez de desarrollar una aplicación web híbrida o en lenguaje HTML 5, el resultado final obtenido es una aplicación real nativa, no distinguible de la que se puede programar con código en *Android Studio*.

  En este TFG se ha utilizado este *framework*.

- ***Expo*** [10] es un *framework* y una plataforma para crear aplicaciones *React Native* universales. En particular, se trata de un conjunto de herramientas y servicios construidos alrededor de *React Native* y plataformas nativas que ayudan al desarrollo, construcción, despliegue y movimiento rápido en iOS Android y aplicaciones web del mismo lenguaje de código que *JavaScript* /*TypeScript*.

  Gracias a *Expo* se puede controlar el progreso de las aplicaciones y ensayar nuevas características a la vez que van desarrollándose. Los programadores tienen la posibilidad de publicar versiones nuevas de las aplicaciones que posteriormente estarán disponibles en los dispositivos a través de *Expo*, y de esta manera, mejorar la calidad de las características que se han ido desarrollando.

  Esta tecnología ha sido utilizada en este TFG en las primeras fases, aunque no ha terminado siendo necesaria para el proyecto final.

- ***Yarn*** [11] es un tipo de instalador de paquetes de *JavaScript* y gestiona dependencias de la aplicación a desarrollar.

  Se ha utilizado *Yarn* para el desarrollo de este TFG.

- ***Node*** [12] es un entorno de ejecución de código abierto basado en *JavaScript*. Es asíncrono.

  Esta tecnología ha sido utilizada en este TFG.

- ***Firebase*** [15] es una plataforma que sirve para el desarrollo de aplicaciones web y móvil, usando una infraestructura de Google. *Firebase* integra funcionalidades para el desarrollo como *Cloud Firestore*, *Firebase ML*, *Cloud Functions*… Está ubicada en la nube y cuenta con Google *Cloud Platform*, la cual utiliza herramientas para crear y sincronizar proyectos. En particular, esta tecnología permite sincronizar, de manera sencilla, los datos de los proyectos sin que haya que administrar la conexión o escribir lógicas de sincronizaciones complejas.

  Firebase provee a los usuarios de sistemas de almacenaje y gestión de recursos para la creación de aplicaciones, ofreciendo tutoriales y consultas gratuitas.

  Las funcionalidades de *Firebase* que se han utilizado en este TFG son *Cloud Firestore*, *Authentication* y *React Native Firebase*. "*Cloud Firestore* almacena y sincroniza la base de datos de la aplicación en la nube", mientras que *Authentication* "autentica usuarios de forma simple y segura". Finalmente, *React Native Firebase* es el dominio en el que poder consultar cómo integrar y utilizar *Firebase* en *React Native*.

- ***GenyMotion***[25] es un emulador de Android para Windows que ha sido utilizado en este proyecto.

- ***VSCode*** [30] (*Visual Studio Code*), es el IDE utilizado en este TFG para desarrollar el código de *React Native*

- ***Docker*** [13] es un proyecto de código abierto que sirve para automatizar el desarrollo de aplicaciones, como si fueran contenedores móviles y autosuficientes que pudieran ejecutarse desde la nube o de manera local. Aunque en un principio se planteó utilizar *Docker* como herramienta de ayuda para realizar los pagos, finalmente, como se explica en *8. 1. Métodos de pago*, no se ha utilizado en este TFG.

- ***Git*** [14] es un sistema de control de versiones (en inglés *VCS, Version Control System*). Este método sirve para mejorar la eficiencia y seguridad en el mantenimiento de aplicaciones que tengan un gran número de archivos de código. La intención de este procedimiento es llevar un control de los archivos compartidos en los cuales se va añadiendo, modificando y eliminando código en un solo ordenador o en varios. *GitHub* [21] permite la colaboración con personas de cualquier parte del mundo realizando una planificación y seguimiento de los proyectos de manera inmediata. Está basado en el sistema *Git*, del cual recoge su control de versiones, mientras que *Hub* es lo que convierte esta línea de comandos en una red social mundialmente reconocida para programadores. Es el repositorio remoto online más grande actualmente.

  El sistema de control de versiones *GitHub,* en particular *Git,* se ha utilizado como repositorio de almacenaje del TFG.

- ***NFC*** (*Near field communication* o comunicación de campo cercano) consiste en una tecnología que autoriza la comunicación inalámbrica y el intercambio de datos entre dos dispositivos que no se encuentren a una distancia mayor de 20cm.

  La tecnología NFC está basada en ISO 14443, o estándares de RFID (identificación de radio frecuencia), por lo que NFC está diseñado para el intercambio de pequeñas cantidades de información.

  La tecnología *contactless*[24] autoriza los pagos con tarjeta a través de la comunicación NFC. Gracias a un chip integrado en el teléfono móvil, se genera un campo electromagnético propio que puede ser susceptible o no al pago por medio de un datáfono.

  De esta forma, el único elemento necesario para realizar un pago con teléfono móvil es que éste tenga tecnología NFC y una app que almacene los datos bancarios necesarios para realizar el pago del importe.

  Este tipo de aplicaciones de pago que acabamos de comentar son las encargadas de verificar la información bancaria y también se aseguran de que la transacción se realiza de manera fiable. Esto da autorización, o no, del pago al teléfono móvil para que la compra tenga lugar.

  A continuación, se presentan las tres formas de pago que se han barajado en este proyecto, las cuales funcionan con tecnología NFC:

- **TPV** [18] (Terminal en Punto de Venta) es un sistema de pago de gestión de ventas para empresas. Ofrece servicio de cobro e impresión de factura simple además de control del *stock*. Para realizar estas operaciones, además de contar con un TPV, es necesario un software de gestión. En tiendas online (*eCommerce*) se puede utilizar TPV Virtual.

  TPV Virtual [19] es un sistema de pago similar a TPV, pero aplicado al mundo web. Ofrece un servicio de gestión del cobro con tarjetas de crédito y débito a través de internet.

- ***Bizum*** [20] es una plataforma (no una aplicación) integrada en las apps bancarias que lo soportan. Gracias a este sistema de método de pago se puede enviar dinero entre particulares de manera inmediata con independencia del banco desde el que realizas esta transacción. Para su utilización es necesario un TPV virtual.

- ***Google Pay*** [3] es una plataforma perteneciente a Google que ofrece un sistema de pago desde dispositivos Android. Permite realizar pagos gracias a NFC transmitiendo toda la información desde la tarjeta bancaria. Es una alternativa a la tarjeta de crédito o débito.

  Más adelante se indicarán los motivos por los que se han descartado TPV Virtual y *Bizum*, para, finalmente, utilizar *Google Pay.*

1. 5. RIESGOS

Como en todos los proyectos, siempre puede haber factores que impidan el desarrollo completo o lo dificulten. En este apartado se ven los principales factores que se han considerado y su posible solución.

RIESGOS DE FUERZA MAYOR

El proyecto a realizar no solo depende de las condiciones en las que está desarrollado, sino que también responde al momento en el que le toca vivir. Como hemos comprobado, un contratiempo como pueda ser el provocado por el COVID-19, frenaría de manera instantánea el avance rápido y correcto del proyecto. De la misma forma, una vez ya acabada, la aplicación podría no ser utilizada durante un posible futuro confinamiento.

Asimismo, pensando en un futuro más lejano, podría ocurrir que el método de pago por datáfonos desapareciera. De hecho, actualmente están comenzando a aparecer nuevas formas de pago, como el reciente *PayPal* con código QR.

En cuanto a la cuestión de la desaparición del pago por datáfono, se podrían incluir otras opciones a las formas de pago de la aplicación. En el caso del COVID-19, supondrían la entrega del proyecto en otra convocatoria posterior.

RIESGOS DE FORMACIÓN

Al haber comenzado este trabajo antes de acabar el curso académico, cabe la posibilidad de que no se superen ciertas asignaturas o que el tiempo requerido para la finalización de las mismas sea mayor de lo esperado.

Por otra parte, al utilizarse lenguajes y tecnologías que no se han utilizado en la Universidad ni en la empresa (como *React Native*, *Git*…), podría suponer una curva de aprendizaje demasiado lenta.

Las soluciones para ambas situaciones supondrían la entrega del proyecto en otra convocatoria posterior, lo cual permitiera la correcta finalización del trabajo. En ambos casos, hay que contar desde un principio con estos riesgos para utilizar un modo de organización que se adelante a estas contingencias.

RIESGO DE INFRAESTRUCTURAS DE DESARROLLO

Como en todo desarrollo informático, al trabajar con librerías, puede ocurrir que las escogidas se actualicen o cambie su forma de uso durante el desarrollo del TFG. Otro problema puede ser que alguna de las funcionalidades a incluir en la app sea de pago o, sin serlo en un principio, lleguen a serlo a posteriori.

En este caso las medidas a adoptar serían, durante el TFG, aislar el uso de las librerías para que sea más fácil su actualización, cambio o mantenimiento, y tenerlo en cuenta estructurando la organización del tiempo de trabajo para poder adaptarse al problema. Si las herramientas necesarias resultan ser de pago, habrá que buscar otras herramientas similares, aunque gratuitas, que cumplan las mismas funciones.

RIESGO PERSONAL

Puede surgir un problema personal que no permita la continuación del desarrollo del proyecto tal y como se señala en la Estructura de Descomposición del Trabajo o EDT del apartado *2.3. Planificación*. Lo importante es intentar que esto no suponga no dedicarle al proyecto el tiempo planificado. Por ello, en este caso, se invertirían fines de semana y/o periodos vacacionales para adelantar el trabajo no realizado por dicho problema personal.

1. 6. METODOLOGÍA

En este proyecto, en el que inicialmente no estaban claros cuáles iban a ser los requisitos finales a cumplir, ya que hubo un cambio de cliente, se propuso una metodología de desarrollo ágil, en la que se pudieran ir modificando los requerimientos según avanzara el proyecto.

Las metodologías ágiles son las que pueden ir adaptándose a la forma de trabajo según las condiciones del proyecto, consiguiendo así una mayor flexibilidad y rapidez en las respuestas para acomodarse al proyecto y su desarrollo. Las empresas que utilizan este tipo de organización son capaces de gestionar sus trabajos de una manera flexible, independiente y eficaz disminuyendo los costes y aumentando la productividad.

Dentro del conjunto de metodologías ágiles, la que se ha escogido es Scrum [22], que es un marco de trabajo para desarrollo de software sujeto a dicho tipo de metodologías. En este tipo de organización de proyectos se aplican una serie de características: la distribución de roles, el reparto del trabajo en equipos y la división del proyecto en Sprints.

En particular, los roles de Scrum son:

- ***Product Owner*:** es el encargado de comprobar que el equipo vaya trabajando de manera adecuada. En este TFG, el *Product Owner* sería Beatriz Pérez.
- ***ScrumMaster*:** se refiere a la persona del equipo que se encarga de que se cumplan los plazos, objetivos y normas del tipo de metodología Scrum. En este TFG, el *ScrumMaster* será Alejandro Ruiz-Olalla.
- ***Equipo de desarrollo***: tienen la responsabilidad de realizar el proyecto. Normalmente son equipos pequeños de entre 3 y 9 personas, entre los que se encuentra también el *ScrumMaster*. En este TFG, el equipo de desarrollo será Alejandro Ruiz-Olalla.
- ***Roles* auxiliares**: serían los clientes, vendedores, etc. Son las personas que consiguen que el proyecto siga adelante y consiguen un beneficio gracias a la realización del mismo. Únicamente participan en el desarrollo del Sprint de captura de requisitos y sus revisiones. En este TFG, el cliente es tanto Beatriz Pérez Valle como Alejandro Ruiz-Olalla, participando en las revisiones principalmente el segundo.

Al tratarse de un Trabajo Fin de Grado, no existe como tal un equipo de desarrollo, ya que es un proyecto individual y, por lo tanto, no se dividen las funciones en grupos de trabajo.

Los Sprints, por otro lado, son períodos de tiempo en los que se distribuyen las tareas, siendo la finalización de cada uno un punto clave en el que se consigue un producto que funciona y se puede enseñar al cliente. Cada Sprint tiene una duración aproximada de 2-3 semanas y se basa en reuniones entre los equipos de desarrollo (en este proyecto hay un único equipo de una sola persona). A continuación, se explicará en cada apartado cómo se ha reflejado este aspecto en la metodología:

- ***Daily Meeting***: reuniones diarias de 15-20 minutos en las que se responden tres preguntas: ¿qué hice ayer?, ¿qué voy a hacer hoy?, ¿qué problemas tuve ayer y cómo voy a solucionarlos? En este TFG, los *daily meetings* son el primer contacto con el proyecto cada mañana, en el que se responden las preguntas escribiéndolas en un cuaderno (*Anexo1*).

- ***Sprint Review***: reuniones de 1 hora y media-2 horas al final de cada Sprint, en las que el equipo revisa el trabajo planificado para ese Sprint que se ha completado y el que no, presenta al cliente el trabajo y propone las guías para el siguiente Sprint. En este TFG, los *Sprint reviews*, se muestran al *Product owner,* y se establece el *Sprint Backlog* (subconjunto de requisitos para el siguiente Sprint).
- ***Sprint Retrospective***: reuniones al principio de cada Sprint en las que se responden tres preguntas: ¿qué fue bien durante el anterior Sprint?, ¿qué fue mal?, ¿qué se puede mejorar para aumentar la productividad? En este TFG, los *Sprint Retrospectives* son momentos de reflexión escritos en un cuaderno (*Anexo1*), y mensajes de correo electrónico con el *Product owner*.

1. 7. REQUISITOS INICIALES

Como se ha adelantado previamente, a la hora de fijar los requisitos de la aplicación, en un primer momento, se mantuvo una reunión con el cliente inicial (Javier Virto), con objeto de concretar los aspectos más relevantes de la aplicación. Esta captura de requisitos se realizó de manera aproximada, ya que no fue posible concretar todos los aspectos que, en primera instancia, el cliente quería para la aplicación. Es por ello, por lo que la lista completa de requisitos se tuvo que definir más adelante durante el desarrollo del proyecto, en lo que hemos denominado Sprint 0. Análisis de aplicaciones, requisitos finales y planificación. En dicho Sprint se decidió, junto con la tutora académica, realizar un análisis de herramientas existentes en el mercado, con un doble objetivo: (1) descartar la existencia de una aplicación similar a la que se pretendía desarrollar y (2) servir como fuente de funcionalidades que podría adoptar nuestra aplicación y, así, servir como apoyo para fijar, junto con los requisitos iniciales proporcionados por Javier, los requisitos finales de nuestra aplicación. Esta diferencia entre los requisitos iniciales y finales es una de las razones por las que se ha escogido Scrum como metodología.

A continuación, se muestran los requisitos (divididos entre funcionales y no funcionales) tal y como quedaron tras esa primera reunión con Javier:

REQUISITOS FUNCIONALES

Los requisitos funcionales establecidos inicialmente fueron los siguientes:

- **Autenticación y registro**. Los usuarios podrán registrarse y autenticarse en la aplicación.
- **Acceso a mapas**. Los usuarios podrán acceder a mapas que muestren los bares en la zona que deseen.
- **Creación de eventos**. La app permitirá crear eventos para tener un “bote” común desde el que poder hacer los pagos en los establecimientos, asimismo se podrá invitar a “amigos” al evento creado.
- **Cobro de eventos**. Como consecuencia directa de “creación de eventos”, el creador del evento decide el dinero que poner de “bote” (por persona) y los participantes reciben un mensaje de solicitud que tienen que aceptar para pagar y poder ser incluidos en el evento.
- **Incorporación al evento**. El usuario administrador del evento puede incorporar a otras personas al mismo. Esto supondrá que se realice una transferencia proporcional a la cuenta actual del evento.
- **Abandono del evento**. Cada participante del evento puede marcharse de éste, siendo abonada a su cuenta la parte proporcional del “bote”.
- **Pagos**. Cobro mediante tecnología *contactless* (NFC) en establecimientos. Esta funcionalidad solamente estará disponible para el administrador del evento.
- **Aumento de dinero**. Posibilidad de envío por parte del administrador de otra solicitud a todos los participantes para añadir más dinero al “bote”. Hasta que todos los participantes no acepten la solicitud, el evento se mantendrá inactivo.

REQUISITOS NO FUNCIONALES

Se pueden organizar en base a lo siguiente:

- **Funcionamiento**. El sistema podrá ser visualizado en cualquier dispositivo móvil que cuente con las propiedades de compatibilidad tanto con la última versión de Google Maps como con la última versión de *Google Pay* y Android.
- **Legislación.** El sistema debe cumplir con las disposiciones recogidas en la Ley Orgánica de Datos Personales y en el Reglamento de Medidas de Seguridad.

Como se ha comentado previamente, dado que los requisitos iniciales que marcó Javier no estaban completamente cerrados, se realizó un análisis de herramientas existentes en el mercado, dando como resultado un listado de requisitos finales de nuestra aplicación, que nos permitiera planificar el desarrollo del proyecto. A continuación, se presentan los resultados obtenidos tras la realización del Sprint 0. Análisis de aplicaciones, requisitos finales y planificación, que se han recogido en dicho análisis.

2. SPRINT 0. ANÁLISIS DE APLICACIONES, REQUISITOS FINALES Y PLANIFICACIÓN

2. 1. ANÁLISIS DE APLICACIONES

En el análisis realizado se han considerado aplicaciones con características similares a las que se pretenden utilizar en ”Pinchapp”, identificando sus semejanzas y diferencias, con objeto de usarlas como punto de partida para establecer las características finales de nuestra aplicación. Las aplicaciones analizadas han sido las siguientes:

- ***Square*** [1] es una aplicación gratuita de punto de venta que proporciona herramientas necesarias para la administración de un negocio (tanto pagos como control del *stock*). Acepta tarjetas tanto de crédito como de débito con un *Square Reader* (lector de tarjetas) que funciona con todas las tarjetas de banda magnética. Los fondos se depositan en poco tiempo en la cuenta bancaria del cliente para que el dinero sea recibido entre uno y dos días.
- ***Momopocket*** [2] es una herramienta de pago. Se accede con una tarjeta de crédito y se crea una cuenta online con la que realizar operaciones electrónicas, tanto compras online como envío de dinero a terceros.
- ***Twyp*** [4] es una aplicación de ING que sirve como plataforma online para realizar los pagos electrónicos de manera segura. Admite pagos *contactless* y se puede añadir a *Google Pay*.
- ***Verse*** [5] es una aplicación con la que enviar y recibir dinero entre particulares. Ofrece control de las transacciones y creación de eventos con grupos de amigos desde la propia app.
- ***Splitwise*** [6] es una herramienta para administrar las cuentas entre amigos, compañeros de piso, etc. Permite dividir gastos y gestionar el presupuesto, además de crear eventos entre compañeros.

A continuación, se muestra la *Imagen 2.1.1,* en la que se han destacado las funcionalidades que se han considerado más importantes de entre las aplicaciones anteriormente indicadas comparándolas con la futura aplicación “Pinchapp”. En dicha tabla se puede ver, para cada herramienta, las funcionalidades a las que da o no soporte, con objeto de poder realizar un mejor análisis de las aplicaciones.

	Square	Momopocket	Twyp	Verse	Splitwise	Pinchapp
Sistema de autenticación	✓	✓	✓	✓	✓	✓
Enviar-Recibir dinero	×	✓	✓	✓	✓	✓
Crear eventos para compartir gastos	×	×	×	✓	✓	✓
Añadir personas al evento ya creado	×	×	×	✓	✓	✓
Eliminar personas al evento ya creado	×	×	×	✓	✓	✓
Mapas	×	×	×	×	×	✓
Añadir más dinero al evento	×	×	×	✓	✓	✓
Gestión del dinero	✓	✓	✓	×	✓	✓
Tablas y gráficos	✓	✓	×	×	✓	×
Método de pago integrados	×	✓	✓	✓	✓	✓
Posibilidad de pagar con NFC	×	×	✓	×	×	✓

Imagen 2.1.1. Análisis de las aplicaciones incluyendo “Pinchapp”.

A continuación, se pasa a analizar las funcionalidades destacadas en la *Imagen 2.1.1*, indicando igualmente si van a ser incluidas en “Pinchapp” o no.

- ***Sistema de autenticación***. Esta característica se refiere a la comprobación de la identidad del usuario, muy importante en operaciones de transferencias y pagos. Esta funcionalidad la comparten todas las aplicaciones y, como era de esperar, se ha decidido que será deseable que “Pinchapp” cuente con sistema de autenticación de correo electrónico y contraseña.
- ***Enviar-recibir dinero***. Funcionalidad que estudia las transferencias entre las cuentas de usuarios de la aplicación, y agiliza el movimiento del dinero. Esta funcionalidad es deseable para la aplicación de “Pinchapp” puesto que la mayoría de las aplicaciones analizadas la tienen ya implementada.
- ***Crear eventos para compartir gastos***. Esta funcionalidad plantea evitar los problemas de no tener efectivo y tener que abonar cada uno parte como parte de un pago común. Ofrece la creación de eventos en los que haya una cuantía común para facilitar el pago entre varios usuarios. Provee de la posibilidad de que una sola persona pague por grupo, de manera que sea más cómodo y rápido de cara al dependiente del establecimiento. A pesar de que la mayoría de las aplicaciones no ofrecen esta funcionalidad (solamente *Verse*), por su propia definición, “Pinchapp” deberá ofrecerla.

- ***Añadir-Eliminar personas al evento ya creado***. Funcionalidad que estudia la entrada y salida de gente a los eventos. Ante la perspectiva de gente que pueda llegar tarde o irse pronto, se puede ir cambiando el número de personas participantes sin tener que eliminar el evento y crear otro. De la misma manera que "*crear eventos para compartir gastos*", a pesar de que la mayoría de las aplicaciones no ofrecen esta funcionalidad, se considera adecuado que "Pinchapp" cuente con ella.
- ***Añadir más dinero al evento***. Se refiere a la posibilidad de añadir dinero al "bote" dentro del mismo evento. Entre las aplicaciones analizadas, por su más estrecha cercanía a "Pinchapp", solamente *Verse* tiene esta funcionalidad. "Pinchapp" deberá ofrecer esta funcionalidad sin límite de operaciones en un mismo evento.
- ***Tablas y gráficos***. Funcionalidad que estudia las formas de visualización de las operaciones y cuentas realizadas en la aplicación. "Pinchapp" no tendría por qué ofrecer esta funcionalidad, ya que el dinero puesto por cada uno de los participantes de un evento es el mismo que el de los demás y el dinero, de sobrar, se reembolsa a cada uno al finalizar.
- ***Mapas***. Esta característica se refiere a la opción de visualización de mapas. Para diferenciarse del resto de aplicaciones, destacar y aparecer más deseable, "Pinchapp" deberá ofrecer esta funcionalidad en la que mostrar los locales donde poder realizar las consumiciones.
- ***Método de pago integrado***. Esta funcionalidad se refiere a la posibilidad de pago desde la aplicación, así como a la realización de transferencias entre los usuarios. "Pinchapp" tendrá que ofrecer esta funcionalidad, incluyendo tanto las transferencias a usuarios como a eventos y pagos en establecimientos.
- ***Posibilidad de pago con NFC***. Funcionalidad que permite el pago en establecimientos. "Pinchapp" deberá ofrecer esta herramienta en todos los lugares que dispongan de datáfono por las propias características de la aplicación.

Como se puede observar en la *Imagen 2.1.1*, "Pinchapp" incluye la gran mayoría de funcionalidades consideradas respecto al resto de aplicaciones, combinándolas y ofreciendo un tipo de aplicación diferente al resto.

2. 2. REQUISITOS FINALES

En base al análisis anterior, y tomando como punto de partida los requisitos iniciales, en este apartado mostramos finalmente los requisitos que hemos establecido para "Pinchapp".

En particular, los requisitos funcionales son los siguientes (para referirme más fácilmente a cada uno de ellos, se han identificado utilizando la notación R*X*):

- **R1. Autenticación y registro**. Los usuarios registrados en la aplicación podrán registrarse y autenticarse en la misma (requisito incluido inicialmente).
- **R2. Amistad entre usuarios**. En este nuevo requisito la aplicación permitirá gestionar una red de "amigos", de forma que los usuarios puedan solicitar "la amistad" de otros y, a su vez, aceptar solicitudes de otros usuarios. De esta manera, los eventos se crearán a partir de los amigos del usuario creador, los cuales constituirán los participantes del evento, facilitando de esta manera la gestión de usuarios participantes en los mismos.
- **R3. Acceso a mapas**. Este requisito permanece igual con respecto a la versión inicial, de forma que los usuarios puedan acceder a un mapa situado en La Laurel. En este mapa se podrán ver los bares que se encuentren en esa localización.
- **R4. Creación de eventos**. La app permitirá crear eventos para tener un "bote" común desde el que poder hacer los pagos en los establecimientos, asimismo se podrá invitar a "amigos" al evento creado (requisito incluido en el listado inicial).
- **R5. Cobro de eventos**. Como consecuencia directa de "creación de eventos", el creador del evento decide el dinero que poner de "bote" (por persona) y los participantes reciben un mensaje de solicitud que tienen que aceptar para pagar y poder ser incluidos en el evento (requisito incluido inicialmente).

- **R6. Consulta de eventos**. Nuevo requisito gracias al cual los usuarios podrán consultar en todo momento el "bote" de los eventos que administran y de los que son participantes.
- **R7. Incorporación al evento**. El usuario administrador del evento puede incorporar a otras personas al mismo. Esto supondrá que se realice una transferencia proporcional a la cuenta actual del evento (requisito incluido inicialmente).
- **R8. Abandono del evento**. Cada participante del evento puede "marcharse" de éste, siendo abonada a su cuenta la parte proporcional del "bote" (incluido inicialmente).
- **R9. Eliminar evento**. El administrador, en este nuevo requisito, puede dar como finalizado un evento. Como resultado, a cada comensal se le devuelve la parte proporcional del "bote", de existir, dividido entre los participantes.
- **R10. Pagos**. Cobro mediante tecnología *contactless* (NFC) en establecimientos (únicamente el administrador de cada evento). (incluido inicialmente)
- **R11. Aumento de dinero**. Posibilidad de envío por parte del administrador de otra solicitud a todos los participantes para añadir más dinero al "bote". Hasta que no acepten todos la solicitud, este evento se mantendrá inactivo (requisito incluido en el listado inicial).

REQUISITOS NO FUNCIONALES FINALES

Estos requisitos no cambian con respecto a los requisitos iniciales.

En base a los requisitos finales, a continuación, se muestra la planificación propuesta para seguir a lo largo del proyecto.

2. 3. PLANIFICACIÓN

En el siguiente EDT (Estructura de Descomposición del Trabajo) se especifican las tareas que se van a realizar en cada uno de los siete Sprints que se han establecido en este TFG, incluyendo el realizado previamente.

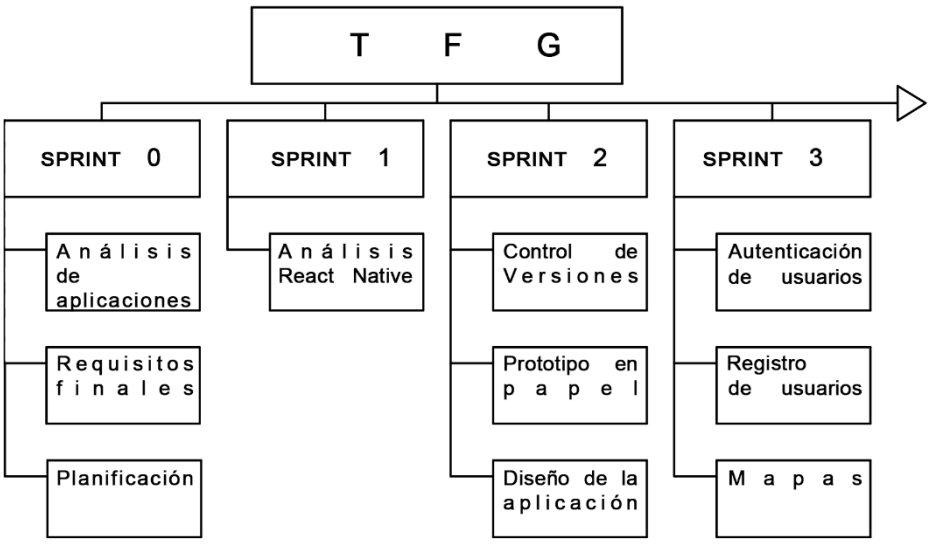


Imagen 2.2.1. EDT de organización de funciones en Sprints considerando los Sprints del 0 al 3.

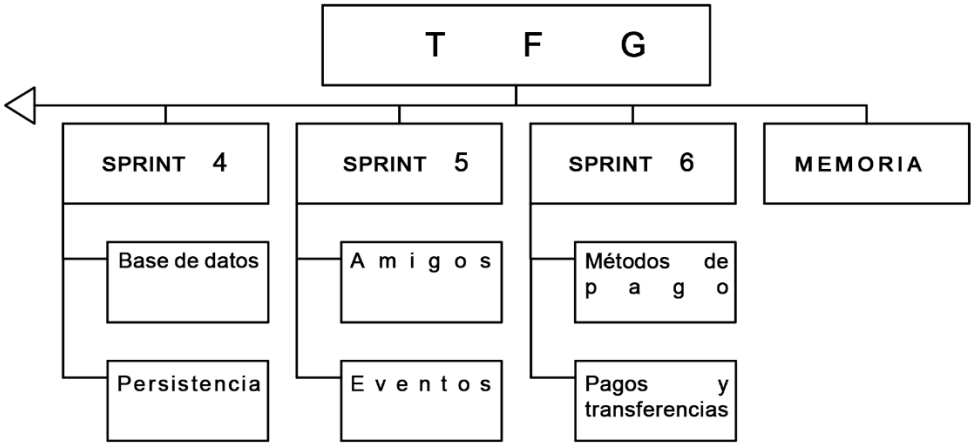


Imagen 2.2.2. EDT de organización de funciones en Sprints considerando los Sprints del 4 al 6 y Memoria.

Nº Actividad	Horas al día de media	Inicio	Final	Días	Horas	semana 3-9 febrero	semana 10-16 febrero	semana 17-23 febrero	semana 24feb-1 marzo	semana 2-8 marzo	semana 9-15 marzo	semana 16-22 marzo	semana 23-29 marzo	semana 30marzo-5abril	semana 6-12 abril	semana 13-19 abril	semana 20-26 abril	semana 27abril-3mayo	semana 4-10mayo	semana 11-17mayo
Sprint 0 Análisis de herramientas requisitos finales y planificación	5,0	03/02/2020	14/02/2020	10	50															
Sprint 1 Análisis React Native	5,0	17/02/2020	13/03/2020	20	100															
Sprint 2 Control de versiones y diseño	2,0	09/03/2020	27/03/2020	15	30															
Sprint 3 Autenticación y mapas	1,5	30/03/2020	10/04/2020	10	15															
Sprint 4 Base de datos	3,0	13/04/2020	24/04/2020	10	30															
Sprint 5 Amigos y Eventos	2,0	20/04/2020	08/05/2020	15	30															
Sprint 6 Pagos y transferencias	2,0	04/05/2020	22/05/2020	15	30															
Memoria	3,0	08/06/2020	12/06/2020	5	15															
TOTAL				100	300															

Imagen 2.2.3. Tabla de organización de tiempo de los Sprints (I).

Nº Actividad	Horas al día de media	Inicio	Final	Días	Horas	semana 18-24mayo	semana 25-31mayo	semana 1-7junio	semana 8-14 junio	semana 15-21 junio	semana 22-28 junio	semana 29junio-5 julio	semana 6-12 julio	semana 13-19 julio	semana 20-26 julio	semana 27julio-2 agosto	semana 3-9 agosto	semana 10-16 agosto	semana 17-23 agosto	semana 24-30 agosto	semana 31 ago-6 septiembre
Sprint 0 Análisis de herramientas requisitos finales y planificación	5,0	03/02/2020	14/02/2020	10	50																
Sprint 1 Análisis React Native	5,0	17/02/2020	13/03/2020	20	100																
Sprint 2 Control de versiones y diseño	2,0	09/03/2020	27/03/2020	15	30																
Sprint 3 Autenticación y mapas	1,5	30/03/2020	10/04/2020	10	15																
Sprint 4 Base de datos	3,0	13/04/2020	24/04/2020	10	30																
Sprint 5 Amigos y Eventos	2,0	20/04/2020	08/05/2020	15	30																
Sprint 6 Pagos y transferencias	2,0	04/05/2020	22/05/2020	15	30																
Memoria	3,0	08/06/2020	12/06/2020	5	15																
TOTAL				100	300																

Imagen 2.2.4. Tabla de organización de tiempo de los Sprints (II).

Teniendo en cuenta el *EDT* anterior, se pueden observar los siguientes Sprints:

SPRINT 0. ANÁLISIS DE APLICACIONES REQUISITOS FINALES Y PLANIFICACIÓN

Como ya se ha visto previamente, en este Sprint se han analizado varias aplicaciones y establecido los requisitos finales. También se han concretado los requisitos finales y se ha organizado la planificación general del trabajo.

Para este Sprint se planificó una duración de 2 semanas.

SPRINT 1. ANÁLISIS DE REACT NATIVE

Se procederá al análisis de *React Native*, en el que se aprenderán los conceptos básicos de este lenguaje y se llegará a un conocimiento avanzado. Se realizarán pruebas de uso de *React Native* para asentar las nociones del lenguaje.

Este Sprint durará 4 semanas.

SPRINT 2. CONTROL DE VERSIONES Y DISEÑO

En primera instancia se desarrollará la organización del control de versiones desde *GitHub*. Además, se comenzará a esbozar un prototipo de la aplicación en papel, a partir del cual realizar el diseño de la misma. También se establecerá el diseño de la arquitectura que se pretende para la aplicación y el diseño de la base de datos.

Este Sprint durará 3 semanas.

SPRINT 3. AUTENTICACIÓN Y MAPAS

Se va a desarrollar el registro, la autenticación, y el acceso a mapas una vez iniciada la sesión como usuario (directamente relacionado con los requisitos R1 y R3). Se integra la funcionalidad de registro y autenticación para poder crear y autenticar, respectivamente, a usuarios. De la misma manera, se implementará la visualización del mapa a través de Google Maps, el cual mostrará los bares que se encuentran en La Laurel.

Este Sprint durará 2 semanas.

SPRINT 4. BASE DE DATOS Y PERSISTENCIA

Se realizarán las labores necesarias para la creación y manejo de la base de datos, siendo esta no relacional. También se desarrollará la parte de la aplicación que interactúa directamente con la base de datos, es decir, la capa de persistencia.

Este Sprint durará 2 semanas.

SPRINT 5. AMIGOS Y EVENTOS

Se desarrollarán todas las funcionalidades referidas a amigos (solicitudes de amistad y confirmaciones), las cuales están directamente relacionadas con el requisito R2. Se diseña también el apartado de eventos, incluyendo desde la creación de los mismos (R4), las solicitudes para ser incluidos en el evento (R4), la incorporación y abandono de participantes (R7 y R8), la consulta de eventos (R6), hasta el aumento de “bote” (R11) y, finalmente, la eliminación del evento (R9).

Este Sprint durará 3 semanas.

SPRINT 6. PAGOS Y TRANSFERENCIAS

Se diseña la capa de presentación necesaria para realizar los pagos y transferencias a “eventos” (requisitos R5 y R10). Se realizarán pruebas de pagos y transferencias a usuarios, integrando una librería que garantice la seguridad en las operaciones y los datos utilizados.

Este Sprint durará 3 semanas.

MEMORIA

La memoria se irá escribiendo al finalizar cada Sprint a modo de consolidación de los conocimientos adquiridos estando estas horas incluidas en la estimación de cada uno. Al finalizar la realización de todos los Sprints, se escribirá la introducción, el seguimiento y control y las conclusiones.

Este Sprint durará 1 semana.

La realización de este TFG, desde el Sprint 0, se compaginará con la asistencia a asignaturas presenciales de la universidad. Por la mañana se acude a la empresa GFI a trabajar en este proyecto (5 horas de lunes a viernes), mientras que por la tarde se cursan dichas asignaturas. A pesar de que en un principio se planeó realizar el proyecto en la empresa, dadas las circunstancias vividas por la pandemia parte del proyecto se tuvo que desarrollar desde casa. En una primera instancia, no se programa trabajar durante los fines de semana ni las vacaciones, aunque no se descarta la posibilidad de que haya que hacerlo en el caso de ser necesario.

3. SPRINT 1. ANÁLISIS DE REACT NATIVE

Una vez analizadas las herramientas y aplicaciones similares a “Pinchapp”, y habiendo dejado fijados los requisitos finales, se procede a investigar el lenguaje *React Native*. En este apartado solo se van a explicar aquellos aspectos del lenguaje que tienen interés para comprender el trabajo realizado. Todo lo demás se refiere a librerías utilizadas para el desarrollo de la aplicación.

Como se ha explicado en la introducción, durante las prácticas de empresa, tuve que aprender el lenguaje *React*, enfocado al diseño de páginas web. Al plantear el TFG se decidió realizarlo mediante el lenguaje *React Native*, que es muy similar a *React*, pero dirigido a aplicaciones móviles, permitiendo desarrollar aplicaciones nativas en iOS y Android.

Gracias al aprendizaje alcanzado de *React*, el estudio de *React Native* supuso una tarea más sencilla de lo que podría haber sido en un principio. Los dos lenguajes se basan en “componentes” (se explicarán con detalle más adelante), los cuales permiten la separación de la interfaz del usuario en piezas independientes y reutilizables y, además, posibilitan que cada pieza sea desarrollada de manera aislada. Conceptualmente estos componentes se asemejan a las funciones con las que se trabaja en *JavaScript*.

Para llegar a comprender *React Native* de manera básica, con objeto de poder comenzar a desarrollar la aplicación, realicé un curso de Edutin Academy [32], “Curso de React Native”. Gracias a este curso, afiancé los conocimientos sobre *JavaScript* y *React*. Durante el mismo, estudié elementos como *Components*, *Props*, *State*, *Style*, o conceptos básicos de *React Native* como pueden ser JSX, cómo hacer *debug*, utilizar librerías como *React Navigation* y empezar a aprender *Expo*.

React Native en vez de separar funcionalidad y visualización, utiliza la maquetación de las páginas web diferenciándolas en intereses. Estos intereses (que se pueden ir reutilizando si el programador así lo requiere) llevan acoplados una serie de “componentes” (de los que hablábamos antes) que pueden contener toda la información (funcionalidad y/o visualización).

Una de las principales diferencias entre *React* y *React Native* es que el segundo utiliza componentes nativos (para aplicaciones móviles) mientras que *React* utiliza componentes web como bloques de construcción. Para comprender completamente *React Native* hay que entender algunos conceptos básicos del lenguaje *React*, como los objetos *state* y *props* o JSX:

- ***Objetos state y prop***. En *JavaScript* hay dos tipos de objetos planos que se denominan *state* y *prop* (properties). Ambos contienen información que contribuye al resultado de la visualización final del código, denominada “render”. A pesar de contener información que afecta a la misma función *render*, *state* y *prop* son diferentes. Mientras que *prop* se introduce en el componente como si fuera un parámetro de una funcion, *state* es administrado dentro del componente, algo parecido a declarar una variable en una función.
- ***JSX***. Es una extensión del lenguaje *JavaScript* orientada al uso de la librería *React*. Es utilizado como preprocesador para transformar el código a *JavaScript,* el cual sea usado como componente en *React*.

```
const element = <h1>Hello, world!</h1>;
```

Imagen 3.1.1. Sintaxis JSX.

- En la *Imagen 3.1.1* podemos observar cómo este tipo de sintaxis no responde a una cadena *string* ni a HTML, sino a un componente JSX.
- JSX produce componentes de *React*. Este tipo de extensiones sirven de azúcar sintáctico para estructurar el código y poder utilizarlo con más eficiencia. Las expresiones JSX se transforman en llamadas periódicas de *JavaScript* hasta que, en última instancia, evalúan un objeto.

Una vez finalizado el aprendizaje del curso, comencé a utilizar *React Native* por mi cuenta con la ayuda del marco de trabajo *Expo* como herramienta para desarrollar en *React Native*, realizando diferentes pruebas para aprender cómo se trabajaba con la herramienta. *Expo* provee de una aplicación móvil con la que poder visualizar el proceso de desarrollo de la aplicación desde el teléfono móvil, por lo que se consideró interesante utilizarla para este proyecto.

Expo CLI es una herramienta diseñada para desarrollar aplicaciones móviles con *Expo*. Para desarrollar en *React Native* hay dos opciones, *Expo CLI* y *React Native CLI*, y para este TFG se decidió utilizar la segunda ya que, como se explicará posteriormente, con *Expo CLI* dio varios problemas que se explicarán posteriormente. Para poder utilizar tanto ésta como *React Native CLI*, otra plataforma de desarrollo de aplicaciones con *React Native*, es necesario preinstalar “Node.js”. “Node.js” es un entorno de *JavaScript* que posibilita ejecutar código en el servidor de forma asíncrona y con un diseño orientado a eventos. De la misma manera, para poder realizar un control de versiones, habría que haber instalado previamente *Git*.

Mientras que *Expo* es un framework y una plataforma para *React Native* universales, "Expo CLI se refiere a la aplicación de línea de comandos que sirve como interfaz entre el desarrollador y las herramientas de *Expo*" [33]. *React Native CLI* es una aplicación de línea de comandos que sirve de interfaz entre el desarrollador y las herramientas de *React Native*. Es decir, cuando son CLI, son interfaces de apoyo a los programadores para el uso del lenguaje en *React Native* y de la plataforma o framework en *Expo*.

Para poder utilizar cualquier tipo de librería propia de *React Native* o creada por los desarrolladores de la comunidad del lenguaje, es necesario instalar la dependencia de *React Native*. De esta manera, al instalarla, la nueva dependencia aparecerá con el nombre y la versión en el archivo “package.json” en el apartado *dependencies*.

Utilizando la plataforma *Expo CLI* se hicieron varias pruebas para testear su eficacia y su uso. Hubo más de un problema con *Expo*, como, por ejemplo, al comienzo de la instalación que normalmente se hace con la instrucción `npm` en el *powershell* del ordenador, pero se bloqueaba continuamente. Por este motivo se procedió a utilizar *`yarn`* `install` (proceso por el cual se instalan las dependencias que están escritas en el archivo “package.json”). Una vez ejecutado el comando `yarn install`, también hubo algún conflicto con la versión de *Node*, ya que hubo que cambiar la versión por otra menos actualizada para que funcionara de manera correcta (v14.4.0)

A la hora de aprender a generar archivos .apk con *React Native* también tuve problemas. Al no ser como la plataforma *Android Studio*, que es mucho más intuitiva, con *Expo* me fue extremadamente difícil generar ese tipo de archivos. Para ello, es necesario configurar el archivo “app.json” del proyecto e introducir los comandos necesarios para generar el archivo .apk estando el *powershell* en la ruta donde está ubicado el proyecto.

```
expo buid:android
```

Imagen 3.1.2. Comando expo

```
yarn expo build:android
```

Imagen 3.1.3. Comando yarn

Cualquiera de estos dos comandos sirve para generar un archivo .apk, pero no conseguí que ninguno de los archivos creados funcionara. Después de varios intentos fallidos, se decidió descartar el uso de *Expo* y se pasó a trabajar directamente con *React Native CLI*.

A partir de este momento, ya no se contaba con las facilidades que proporcionaba *Expo* para el desarrollo en *React Native* (como la visualización, por ejemplo). Por esta razón, se tuvo que buscar otra manera de visualizar los cambios realizados en el código sin tener que generar el archivo .apk a cada momento, ya que es un proceso costoso en el tiempo. El primer emulador que se probó fue en *Android Studio*, ya que en la asignatura *Informática Móvil* se estudió esta herramienta de desarrollo. Sin embargo, hubo varios problemas al intentar configurarlo como emulador predeterminado del sistema operativo y se descartó esta posibilidad. Es por ello por lo que tuve que hacer una búsqueda de emuladores compatibles con *React Native* y encontré *GenyMotion*, que utiliza *VirtualBox* [7] para simular una conexión vía USB con el ordenador.

Se consiguió configurar las variables de entorno del sistema Windows para asociar el emulador de *GenyMotion* como emulador por defecto para ejecutar aplicaciones de *React Native*. Se realizaron varias pruebas para comprobar su funcionamiento y, al ser positivas, se definió este emulador como el definitivo.

Una vez concretada la forma de trabajo y establecido el emulador, se procedió a la búsqueda de librerías relacionadas con las funcionalidades y requisitos de la aplicación para su posterior uso en su desarrollo.

- ***react-native-navigation***[26]. Para la navegación dentro de la aplicación se decidió utilizar esta librería tras comprobar que su utilización era sencilla e intuitiva. Esta librería requiere de *react-native-gesture-handler* para ciertas funcionalidades, como, por ejemplo, el uso de tecnología táctil.
- ***react-native-gesture-handler***. Una librería de gestión de las APIs, para ofrecer la navegación en la app mediante la tecnología táctil.
- ***react-native-firebase***. Librería necesaria para su uso como *backend*, o sistema de almacenaje de base de datos. Inicialmente no se sabía si iba a ser necesario el sistema de autenticación sin necesidad de usar una base de datos. Había dudas de si habría que utilizarla para consultar los datos del usuario y contraseña como algo separado de la base de datos. Posteriormente se descubrió que era necesario instalar una dependencia para la conexión con *Firebase* desde “Pinchapp”, otra librería para la autenticación y otra más para la base de datos *(@react-native-firebase/app, @react-native-firebase/auth, @react-native-firebase/firestore*, respectivamente).
- ***react-native-maps***[27]. Para la visualización de mapas generados por la aplicación Google Maps, junto con muchas de las funcionalidades propias de la librería (ubicación, información, locales…)
- ***Bizum***. Frente a la problemática de cómo integrar las funciones de pago, en un primer momento se pensó en utilizar librerías para poder integrar *Bizum*. Se envió un correo electrónico a la empresa correspondiente, pero la respuesta fue un enlace a una página web en la que explicaba la forma de uso de este tipo de métodos de pago, para lo cual era necesario disponer de un TPV virtual. Se intentó conseguir un TPV virtual, pero tras hablarlo con varias entidades bancarias, se descartó por ser de pago. La opción elegida finalmente para esta funcionalidad se explicará de manera pormenorizada en *8. Sprint 6. Pagos y transferencias.*

4. SPRINT 2. CONTROL DE VERSIONES Y DISEÑO

En este Sprint, se abordan los temas relacionados con el control de versiones, con el prototipado de la aplicación, y con el diseño de la arquitectura y de la base de datos de la misma.

4. 1. CONTROL DE VERSIONES

Como ya se ha explicado previamente, el controlador de versiones que se utilizará en este proyecto será la parte *Git*, y *GitHub* como almacenador de código, que a su vez estará administrado por *Git*.

El trabajo en *Git* no sólo se almacena de manera incremental, sino que va guardándose en momentos de trabajo. Cada vez que se confirma un cambio (*commit*), *Git* guarda una copia del trabajo que hay creado en ese momento, con los metadatos del autor y un mensaje explicativo.

Una rama *Git* es el directorio donde se almacenan las referencias a cada una de esas confirmaciones (*commits*). Al iniciar *Git*, se crea por defecto una primera rama *Master* y, a partir de ahí, se pueden ir creando otras ramas principales. Al final de cada Sprint, la rama creada en el mismo se une a la rama principal del trabajo gracias a una unión (*merge*) y, de esta forma, se guarda como un solo momento de trabajo.

Se crearán dos ramas principales:

- *Máster*. En la que se guarde el proyecto final.
- *Develop*. En la que se vayan uniendo las ramas de cada Sprint. A partir de ésta se irán creando el resto de las ramas Sprint 2, Sprint 3…

4. 2. PROTOTIPADO

En esta sección se presenta el análisis de flujo de los pasos que conllevaría el uso de la aplicación, según la funcionalidad que ofrece al usuario.

En las *Imágenes 4.2.1* y *4.2.2* se muestra, siguiendo el estilo "*story board*", una sucesión de las posibles pantallas de las que estaría formada la aplicación, numeradas consecutivamente, simulando el uso de la misma.

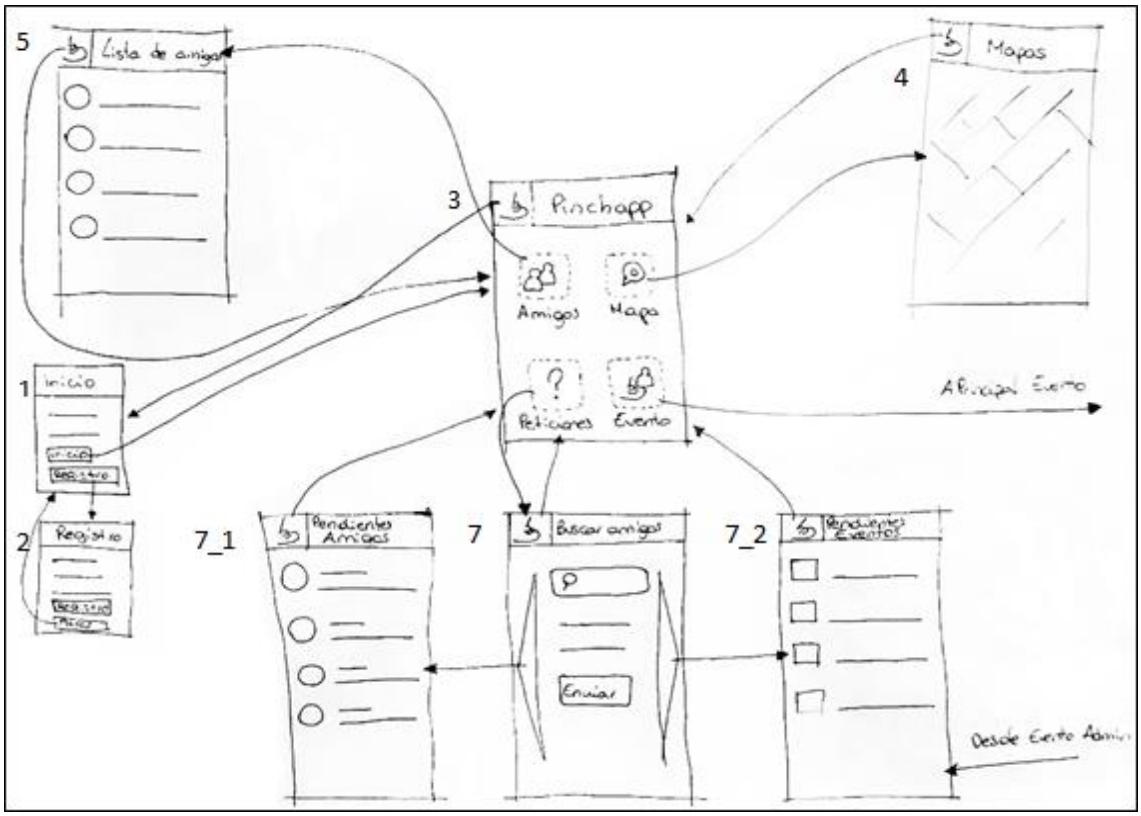


Imagen 4.2.1. Esquema de flujo de “Pinchapp” (I)

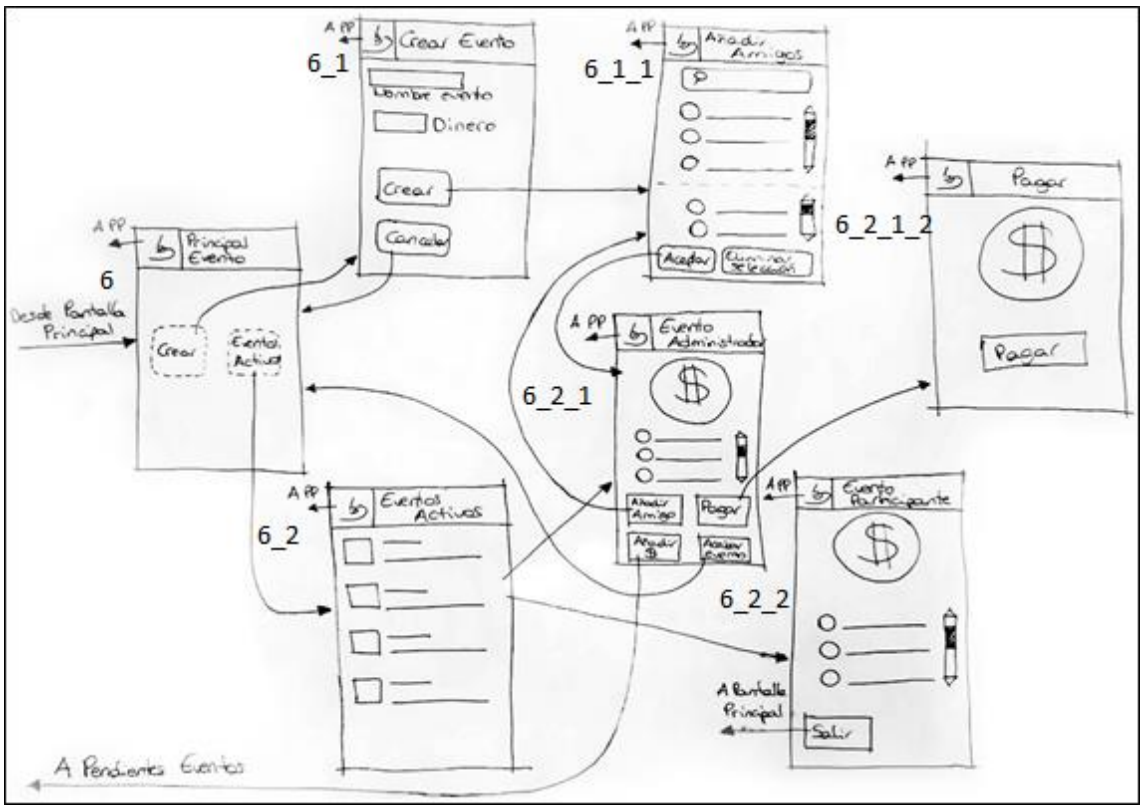


Imagen 4.2.2. Esquema de flujo de “Pinchapp” (II)

Siguiendo el esquema del prototipo, que no tiene por qué ser el resultado final, aunque si una orientación muy cercana, se han ido nombrando las pantallas. A continuación, se va a explicar el flujo que siguen las imágenes siguiendo el orden numérico del "*story board"*:

- **1_Inicio**. Como se puede observar, al inicio de la aplicación aparece una pantalla en la que poder realizar dos acciones, iniciar sesión o acceder a otra pantalla para registrarse. Para iniciar sesión hay dos cuadros de texto en los que introducir en uno el correo electrónico y en otro la contraseña y un botón *Iniciar Sesión,* que clicar para realizar la acción y navegar a "3_Pantalla Principal Pinchapp". Si por el contrario se quiere hacer un nuevo usuario se pulsará el botón *Registrarse* y se llegará a "2_Registro".
- **2_Registro**. En esta pantalla habría 3 cuadros de texto, uno en el que introducir el nombre, otro en el que introducir el correo electrónico y otro en el que escribir la contraseña, y dos nuevos botones, uno para crear el nuevo usuario *Registrarse* y otro para volver a la pantalla anterior *Atrás*.
- **3_Pantalla Principal "Pinchapp"**. Una vez que nos hemos logueado en la aplicación, en esta primera pantalla el usuario tiene cuatro iconos:
    - Mapa. Gracias a un botón *Mapa*, el usuario puede acceder a la pantalla "4_Mapa".
    - Amigos. La segunda opción del menú permite acceder a "5_Amigos".
    - Solicitudes. Lleva al usuario a la pantalla "7_Solicitudes".
    - Evento. Permite acceder a la pantalla "6_Evento".
- **4_Mapa**. Se puede observar un mapa ubicado en La Laurel, por en el que poder navegar.
- **5_Amigos**. Muestra un listado de usuarios amigos o, en su defecto, un mensaje de alerta informando de que no tiene amigos.
- **6_Evento**. Pantalla con dos botones, uno llamado *Crear*, que lleva a la pantalla "6_1_Crear Evento", y otro llamado *Eventos Activos*, con el que se accede a "6_2_Eventos Activos".
    - **6_1_Crear Evento.** En esta pantalla aparecen dos cuadros de texto y dos botones. En los cuadros de texto hay que introducir un nombre para el evento y una cantidad de dinero que pedir a cada uno de los participantes del evento. Los botones incluidos en esta pantalla son *Añadir comensales*, que lleva a la pantalla "6_1_1_Añadir comensales", y otro llamado *Cancelar,* que devuelve a la pantalla "6_Evento".
        - **6_1_1_Añadir comensales.** Aparece un listado de los amigos del usuario que todavía no hayan sido invitados al evento y un botón debajo de cada uno que permite invitarlos al mismo. A su vez, en la pantalla hay dos botones, uno llamado *Crear evento,* que crea el evento y devuelve al usuario a la pantalla "6_Evento", y otro llamado *Cancelar*, que lleva a la pantalla "6_Evento" sin haber creado el evento.
    - **6_2_Eventos Activos.** Ofrece dos listados de eventos, uno en el que se muestran los eventos activos en los que el usuario es el administrador (quien ha creado el evento) y otro en el que se muestran aquellos eventos activos en los que participa el usuario sin ser el creador. Al hacer clic en cada uno de los elementos de las listas, la aplicación lleva a la pantalla "6_2_1_Evento Administrador" o a "6_2_2_Evento Participante", respectivamente. En caso de no haber eventos activos, aparece un texto avisando de ello. En esta pantalla aparece también un botón llamado *Atrás*, que devuelve a la pantalla "6_Evento".

- **6_2_1_Evento Administrador.** Aparece el dinero total que queda asociado al evento, una lista de los comensales y cinco botones:
  - *Añadir comensales*. Al hacer clic lleva al usuario a la pantalla "6_1_1_Añadir comensales".
  - *Acabar evento*. Finaliza el evento y devuelve al usuario a "6_2_Eventos Activos".
  - *Aumentar Bote*. Lleva a la pantalla "6_2_1_1_Aumentar Bote".
  - *Atrás*. Devuelve a la pantalla "6_2_Eventos Activos".
  - *Pagar*. Lleva a la pantalla "6_2_1_2_Pagar".
    - **6_2_1_1_Aumentar Bote.** Ofrece un cuadro de texto en el que introducir la cantidad a pagar de nuevo por cada uno de los miembros del grupo y dos botones, *Aumentar* que realiza una nueva petición a evento a todos los comensales y lleva a la pantalla "6_2_Eventos Activos" y *Cancelar*, que devuelve al usuario a la pantalla "6_2_1_Evento Administrador".
    - **6_2_1_2_Pagar.** Se llega desde la pantalla "6_2_1_Evento Administrador" y se lleva a cabo el pago a través del teléfono móvil. Gracias al botón *Atrás*, se vuelve a "6_2_1_Evento Administrador" y la cantidad de "bote" habría disminuido según el pago.
- **6_2_2_Evento Participante.** Aparece el dinero total que queda en el evento, una lista de los comensales y dos botones:
  - *Salir de evento*. Elimina al usuario de la lista de participantes del evento y lo devuelve a "6_2_Eventos Activos".
  - *Atrás*. Devuelve a la pantalla "6_2_Eventos Activos".

- **7_Solicitudes**. Es una pantalla simple en la que poder enviar solicitudes de amistad a otros usuarios mediante un cuadro de texto en el que introducir el email del usuario en cuestión y un botón para realizar el envío de la solicitud. En esta pantalla hay a su vez otros dos botones, uno que lleva a la pantalla "7_1_Solicitudes Amistad" y el otro que lleva a "7_2_ Solicitudes Eventos".
  - **7_1_ Solicitudes Amistad.** Ofrece dos listados, uno en el que aparecen las "solicitudes de amistad pendientes de aceptar" y otra de las "solicitudes de amistad enviadas". Cada uno de los elementos de la lista "solicitudes pendientes de aceptar" tiene asociados un botón de *aceptar* y otro de *cancelar*.
  - **7_2_ Solicitudes Eventos.** Ofrece dos listados, uno en el que aparecen las "solicitudes de eventos pendientes de aceptar" y otra de las "solicitudes de eventos enviadas". Cada uno de los elementos de la lista "solicitudes pendientes de aceptar" tiene asociados un botón de *aceptar* y otro de *cancelar*.

4. 3. DISEÑO

En este apartado presentaremos cuál es el diseño de la arquitectura elegida para la aplicación, y describiremos con más detalle el diseño de la base de datos.

En lo que refiere a la arquitectura, se va a utilizar una arquitectura basada en capas. La programación por capas tiene como principal objetivo la separación (desacoplamiento) de las partes que componen un sistema software. Quizá, la estrategia más empleada es utilizar tres capas: *presentación*, *lógica de negocio* y *persistencia*. En este proyecto se va a utilizar una programación en dos capas ya que *React Native* invita al programador a unir la capa de presentación y de lógica de negocio en una única capa llamada "presentación y lógica".

En cuanto a la base de datos, se va a utilizar una base de datos no relacional. Como ya se explicó en el apartado *1. 4. Tecnologías*, la gestión del almacenamiento se va a realizar mediante *Firebase,* ya que, entre otras cosas, permite el almacenamiento de la información y cuenta con una interfaz gráfica en la que ver a tiempo real todos los cambios que se realizan en la base de datos.

Firebase cuenta con varias formas de almacenar información en la nube como por ejemplo *Cloud Firestore* o *Firebase Realtime Database*. Para este proyecto se ha decidido utilizar *Cloud Firestore*, una base de datos NOSQL, entre otros motivos, porque ha sido el sistema del que más información se ha encontrado en internet. Este tipo de bases de datos contienen *colecciones* (semejante a clases) que a su vez contienen *documentos* (similar a objetos), dentro de los cuales se pueden incluir *campos* (como si fueran atributos de un objeto). Estos campos pueden ser a su vez documentos, de tal manera que un documento puede contener otros documentos.

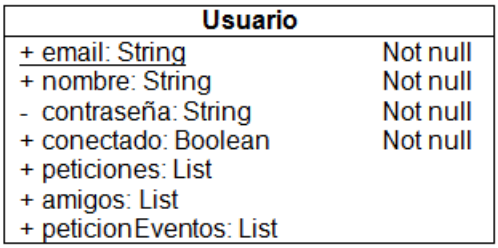


Imagen 4.3.1 Colección usuario

Para la realización de este proyecto, se van a crear dos colecciones principales: “usuarios” y “eventos”. Estas dos colecciones van a contener documentos que todavía no van a ser designados ya que se crearán desde la aplicación cuando ésta esté en uso. El esbozo de cómo se van a diseñar “usuarios” sería el siguiente. Para cada usuario de la aplicación se generará un documento, cada uno de los cuales se compondría de los siguientes campos:

- ***Email***. Sería la clave que distingue a los usuarios entre sí. Se entiende que cada usuario tiene una dirección de correo electrónico único, de ahí que se utilice como identificador. Este tipo de campo sería obligatorio en cualquier documento de “usuarios”.
- ***Contraseña***. Se guardarían los datos de la contraseña cifrada para poder comprobar la veracidad del usuario. Este tipo de campo sería obligatorio en cualquier documento de “usuarios”.
- ***Nombre***. Sería un campo cuyo valor puede repetirse en varios usuarios. Este tipo de campo sería obligatorio en cualquier documento de “usuarios”.
- ***Conectado***. Campo *booleano* que tendría valor *true* si el usuario está conectado y *false* en caso contrario. Este tipo de campo sería obligatorio en cualquier documento de “usuarios”.
- ***Peticiones (o solicitudes)***. Para un usuario A, este campo tipo *map* contendrá un *array* con un conjunto de valores *clave-valor* en el que las claves son generadas de manera aleatoria y cada valor corresponde al identificador del usuario, es decir, a su email, al que el usuario A ha solicitado ser su amigo (pero que todavía se desconoce si dicha petición se ha aceptado o no). Este tipo de campo no sería obligatorio en los documentos de “usuarios”.
- ***Amigos***. Para cada usuario A, este campo tipo *map* contendrá un *array* con un conjunto de valores *clave-valor* en el que las claves son números sucesivos y en el que cada valor corresponde al email de los usuarios que hayan aceptado previamente una petición de amistad enviada por el usuario A. Este tipo de campo no sería obligatorio en los documentos de “usuarios”.
- ***PeticiónEventos***. Este campo registrará las solicitudes que cada usuario creador de un evento realiza, para solicitarles a otros usuarios que participen en el evento. Es un campo tipo *map* que contendría un conjunto de valores *clave-valor* en el que la clave es generada de manera aleatoria y el valor es otro *map*. Este segundo *map* contiene dos campos *clave-valor*, donde el primero sería “nombreEvento” que contiene el “IDEvento” del evento desde el cual se envía la petición, y el segundo “dinero” que sería el precio que los futuros participantes al evento tendrían que pagar al aceptar la invitación. Este tipo de campo no sería obligatorio en los documentos de “usuarios”.

Evento	
+ idEvento: String	Not null
+ nombreEvento: String	Not null
+ emailAdmin: String	Not null
+ dinero: Double	Not null
+ eventoActivo: Boolean	Not null
+ noConfirmados: List	
+ confirmados: List	

Imagen 4.3.2. Colección evento

El diseño general de la categoría “eventos” quedaría de esta forma:

- ***iDEvento***. Este campo sería un *string* compuesto por: “nombreEvento”_”emailAdmin”. Este tipo de campo sería obligatorio en los documentos de “eventos”.
- ***NombreEvento***. Es el *string* que contendría el nombre del evento y es asignado por el administrador. Este tipo de campo sería obligatorio en los documentos de “eventos”.
- ***EmailAdmin***. Sería el identificador del usuario que ha creado el evento. Este tipo de campo sería obligatorio en los documentos de “eventos”.
- ***Dinero***. Sería la cantidad total asociada al evento. Este tipo de campo sería obligatorio en los documentos de “eventos”.
- ***EventoActivo***. Campo *booleano* que se encontraría en valor *true,* si todos los participantes del evento se encuentran en el campo “confirmados”, y adquiere valor *false,* si existe el campo “noConfirmados”. Este tipo de campo no sería obligatorio en los documentos de “eventos”.
- ***NoConfirmados***. Es un campo tipo *map* que contendría un *array* con un conjunto de campos *clave-valor* en el que las claves son generadas de manera aleatoria y el valor es el identificador de cada usuario que aún no habría confirmado la solicitud para unirse a dicho evento. Este tipo de campo no sería obligatorio en los documentos de “eventos”.
- ***Confirmados***. Es un campo tipo *map* que contendría un *array* con un conjunto de campos *clave-valor* en el que las claves son generadas de manera aleatoria y el valor es el identificador de cada usuario que haya confirmado la solicitud para unirse a dicho evento. Este tipo de campo no sería obligatorio en los documentos de “usuarios”, pero su existencia dependería del campo “noConfirmados”, es decir, si “noConfirmados” no existiera, “confirmados” debería existir y viceversa. En el caso de haber usuarios que hayan confirmado la solicitud a un evento dado y, a su vez, usuarios que todavía no hayan confirmado dicha solicitud, ambos campos existirían a la vez.

5. SPRINT 3. AUTENTICACIÓN Y MAPAS

Aunque a partir de este Sprint se vayan a mostrar algunas de las pantallas finales de la aplicación “Pinchapp”, todas las imágenes tanto de la composición gráfica en áreas de acción como las finales de la visualización de la aplicación se encuentran en el Anexo 2.

5. 1. NAVEGACIÓN

Antes de comenzar el trabajo de la funcionalidad comprendida, en particular, en este Sprint, se creó un repositorio en *GitHub* llamado “Pinchapp”. También se creó una carpeta donde ir almacenando todo el código de manera local y mediante *powershell* (ejecutado como administrador y estando en la ruta de la carpeta en la que crear el proyecto), ejecutamos “`react-native init nombre de la aplicación`” (en este caso de nombre se escogió “Pinchapp”), con objeto de crear el proyecto.

Una vez creado el proyecto y teniendo instalado *VSCode* se añade una extensión de *Git* y se realiza el primer “*commit and push*” hacia el nuevo repositorio, gracias a una clave remota proporcionada por *GitHub*. De esta forma se genera la primera instancia del proyecto en el repositorio de *GitHub*. A continuación, se crea una rama en *GitHub* llamada *develop* que, tal y como se ha dicho previamente, es donde se irán guardando los resultados de todos los Sprints. Se crea a su vez otra rama desde *Git* llamada Sprint 2 y se comienza a implementar la librería *react-native-navigation* y toda la navegación.

Como ya se ha explicado en el “Sprint1.Análisis de *React Native*”, para implementar la navegación entre las pantallas de la aplicación se ha decidido utilizar *react-native-navigation*.

Para utilizar *react-native-navigation* es necesario instalar estas librerías:

- Las librerías necesarias para poder utilizar la dependencia *de react-native-navigation: @react-navigation/native, @react-navigation/stack*
- Las recomendadas desde la página oficial de *React Native* para evitar posibles fallos y problemas con ésta y otras librerías son: *react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view*

Todas estas librerías es necesario configurarlas de manera manual o, si se utiliza una versión posterior a la v0.60 de *React Native*, se puede realizar también ejecutando el comando “`react-native link +nombre de la referencia a ejecutar`”

Para la implementación de la navegación es necesario crear una clase .js para cada uno de los componentes, tanto de las pantallas como de la administración de las mismas.

```
26  const Stack = createStackNavigator();
27
28  function AdministrarPantallas() {
29    return (
30      <NavigationContainer ref={navigationRef}>
31        <Stack.Navigator initialRouteName="Logging">
32          <Stack.Screen name="Logging" component={PantallaLogin}
33            options={{headerShown: null}}/>
34          <Stack.Screen name="Registro" component={PantallaRegistro}
35            options={{headerShown: null}}/>
36          <Stack.Screen name="PantallaPrincipal" component={PantallaPantallaPrincipal}
37            options={{headerShown: null}}/>
38          <Stack.Screen name="VerMapa" component={PantallaVerMapa}
39            options={{headerShown: null}}/>      You, a few seconds ago • Uncommitted
40        </Stack.Navigator>
41      </NavigationContainer>
42    );
43  }
44
45  export default AdministrarPantallas;
```

Imagen 5.1.1. Componentes de pantallas 1_Inicio, 2_Registro, 3_Pantalla Principal “Pinchapp” y 4_Mapa en index.js.

Como se puede observar en *Imagen 5.1.1*, cada “*component*” dentro de la etiqueta “Stack.Screen” es una clase .js con su propia funcionalidad y contiene la visualización de la pantalla asociada.

Para que cada pantalla pueda funcionar, es necesario que esté integrada en un componente “NavigationContainer”. Mediante este componente se le asigna un valor a “navigationRef” pudiendo de esta manera acceder a los datos de esta navegación desde el componente principal (“PantallaLogin.js”).

Todos los componentes “Stack.Screen” deben estar envueltos en “Stack.Navigator” para poder navegar entre las pantallas del interior de la aplicación. En “Stack.Navigator” se puede agregar una *prop* “initialRouteName” a la que asignar el nombre (*name*) de una de las “Stack.Screen” que se vea la primera en la aplicación como se ve en la línea 31 de la *Imagen 5.1.1*. Dentro de “Stack.Navigator” se crean tantos “Stack.Screen” como pantallas sean necesarias. En “Stack.Screen” se pueden asignar varias *props* diferentes:

- ***name***. El nombre que se asigna a *name* en un componente de “Stack.Navigator” será el nombre al que se hará referencia para navegar hacia esa pantalla.
- ***component***. El valor que se pone en *component* es el nombre de la clase que se importa del archivo .js que lleva el mismo nombre que la clase que se ha importado a “index.js”.
- ***options***. Proporciona una serie de opciones que aplicar a la pantalla tanto al menú superior como a las opciones que aparecen en él. Por ejemplo, en las líneas 33, 35, 37 y 39 de la *Imagen 5.1.1* si la opcion “headerShown” está marcada como *null*, la cabecera que sale por defecto en las pantallas, no aparecería.

```
1   import * as React from 'react'
2   import {View, TextInput} from 'react-native'
3   import BotonPersonal from '../AdministrarPantallas/BotonPersonal'
4   import * as RootNavigation from '../AdministrarPantallas/RootNavigation'
5
    Unsaved changes (cannot determine recent change or authors)
6   class PantallaLogin extends React.Component {
7     render(){
8       return (
9         <View style={{flex: 7, alignItems: 'center', justifyContent: 'center', borderWidth: 0,
10          borderColor: 'teal', backgroundColor: '#fff'}}>
11          <TextInput style={{height: 30, borderColor: 'black', borderWidth: 180}} />
12          <TextInput style={{height: 30, borderColor: 'black', borderWidth: 180}} />
13          <BotonPersonal colorFondoBoton='#fff' colorTextoBoton='orange'
14            nombre='Registrarse' accion={() => RootNavigation.navigate('Registro')}/>
15          <BotonPersonal colorFondoBoton='#fff' colorTextoBoton='orange'
16            nombre='Login' accion={() => RootNavigation.navigate('PantallaPrincipal')}/>
17        </View>
18    )
19    }
20  }
21
22  export default PantallaLogin
```

Imagen 5.1.2. Métodos pantalla 1_Inicio

Como se puede observar en *Imagen 5.1.2* esta clase cuenta con un método *render* (línea 7 de la *Imagen 5.1.2*) dentro del cual hay un *return* (línea 8 de la *Imagen 5.1.2*) que devuelve una estructura de etiquetas como las que se pueden ver en la creación de páginas web. Mediante estas etiquetas se genera la visualización de la app.

Desde “index.js”, al llamar a esta clase en el “Stack.Screen” se le ha puesto una *prop* “*options={{headerShown: null}}*” (ver línea 33, 35, 37 en la *Imagen 5.1.1*) para que no aparezca la cabecera de navegación de la app. Como se puede comprobar, en las dos acciones incluidas como *props* en “BotonPersonal” (ver líneas 14 y 16 en la *Imagen 5.1.1*) se utiliza la clase “RootNavigation” para poder utilizar los métodos “*navigate”* de la librería *react-native-navigation*. Con estos métodos se puede navegar a una pantalla que tenga el nombre indicado anteriormente en *name*, pasando este nombre (*string*) como argumento a la función “*navigate”*.

La visualización de esta pantalla sería la *Imagen 5.1.3*.

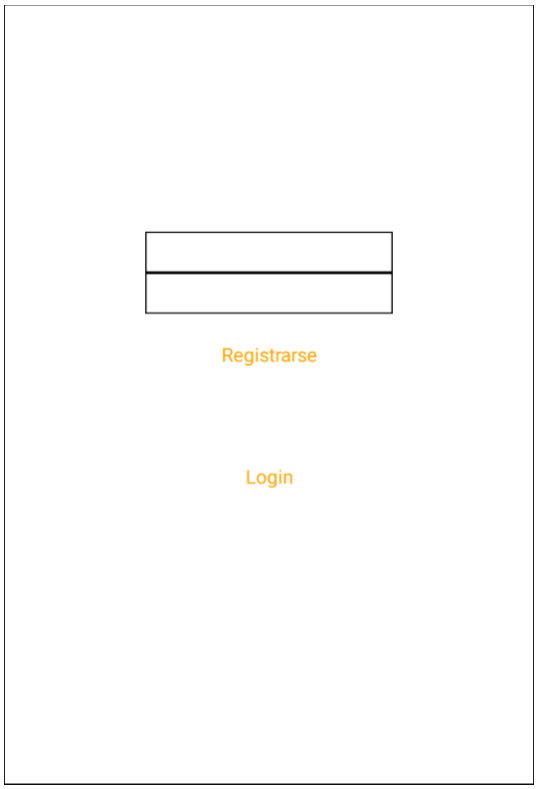


Imagen 5.1.3. Pantalla 1_Inicio Imagen con *headerShown* siendo *null*

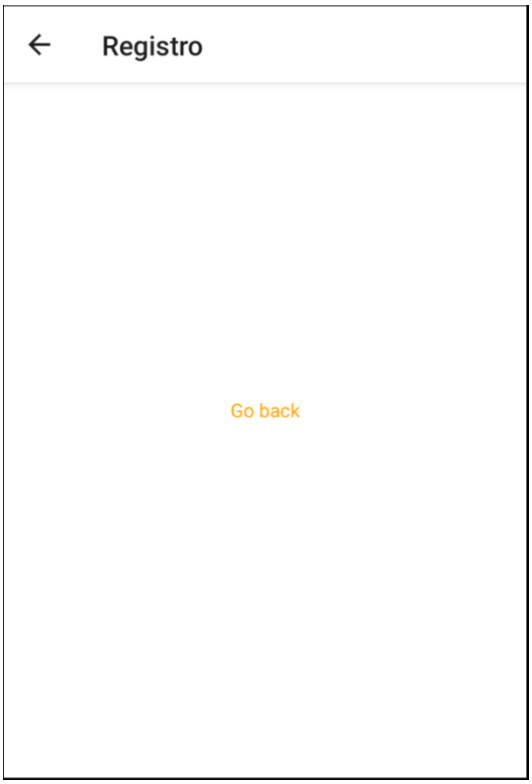


Imagen 5.1.4. Pantalla 2_Registro sin *headerShown* siendo *null*

```
1   import React from 'react'
2   import {View} from 'react-native'
3   import BotonPersonal from '../AdministrarPantallas/BotonPersonal'
4   import { NavigationContext } from '@react-navigation/native';
5
    You, a few seconds ago | 1 author (You)
6   class PantallaRegistro extends React.Component {
7     static contextType = NavigationContext;
8     render(){
9       const navigation = this.context;
10      return (
11        <View style={{ flex: 7, alignItems: 'center',justifyContent: 'center', borderWidth: 0, borderColor: 'teal', backgroundColor: '#fff'}}>
12          <BotonPersonal colorFondoBoton='#fff' colorTextoBoton='orange' nombre='Go back' accion={() => navigation.goBack()}/>
13        </View>
14      )       You, 2 months ago • Primer comit
15    }
16  }
17
18  export default PantallaRegistro
```

Imagen 5.1.5. Métodos Pantalla 2_Registro

La *Imagen 5.1.5* hace referencia a la clase “Pantalla 2_Registro.js”. Como se puede observar en dicha imagen, el método “*navigate”* (ver línea 12 en la *Imagen 5.1.5*) se utiliza de manera diferente a como se ha utilizado previamente, En esta situación, el método “*navigate*” se obtiene del objeto “*navigation*” (ver línea 9 en la *Imagen 5.1.5*) que se extrae de una serie de operaciones con el objeto “NavigationContext” extraido de @react-navigation/native (ver línea 4 en la *Imagen 5.1.5*). En este caso, para navegar a la pantalla anterior (PantallaLogging) en vez de utilizar “navigation.navigate(“Logging”)” se utiliza “navigation.goback()” ya que el método se refiere de manera directa a la pantalla anterior. Para este archivo no se han añadido ninguna *prop* en las *options* de “Stack.Screen” (ver línea 33, 35, 37 en la *Imagen 5.1.1*), por eso se puede ver en la cabecera de la *Imagen 5.1.4*.

5. 2. AUTENTICACIÓN Y REGISTRO

Como se ha explicado en *1.4. TECNOLOGÍAS*, al utilizar *Firebase*, esta plataforma, también provee de *Firebase Authentication*, que proporciona servicios *backend* fáciles de utilizar con *React Native*. *Firebase Authentication* provee de una interfaz gráfica mediante la cual se pueden realizar varias tareas. En particular, en la siguiente imagen mostramos la pantalla correspondiente a Firebase Autentication, de nuestra cuenta de Firebase (console.firebase.google.com).

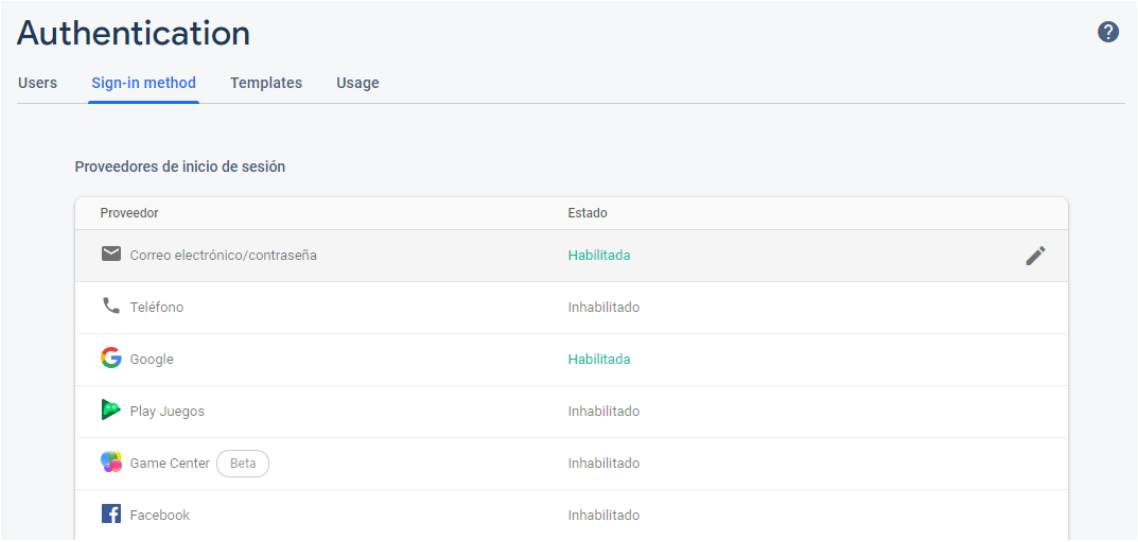


Imagen 5.2.1. *Firebase Authentication*

Como se puede observar en la *Imagen 5.2.1*, desde esta interfaz de Firebase se pueden habilitar diferentes métodos de autenticación (correo electrónico, número telefónico, Google…).

Además, se pueden comprobar los usuarios que se crean, conocer la fecha de creación, el ID, el UID de usuario, así como mediante qué método se han autenticado. También se pueden crear de manera manual nuevos usuarios desde esta plataforma web y obtener los parámetros *hash* de las contraseñas. En la pestaña *Usage* de esa pantalla, se puede consultar el uso de la autenticación en la aplicación.

A continuación, se va a proceder a la explicación de la integración de esta tecnología en la aplicación y cómo se ha utilizado.

Como se ha ido explicando a lo largo de esta memoria, se van a utilizar varias herramientas de *Firebase*, por lo que lo primero que hemos tenido que hacer ha sido agregar *Firebase* al proyecto “Pinchapp” creado en el IDE *VS Code* para poder utilizarlo. Para ello, hay que crear previamente un proyecto desde Firebase (la interfaz gráfica explicada anteriormente). Más tarde habría que instalar la dependencia de *Firebase* en el proyecto creado en el IDE *VS Code*, para lo cual son necesarios ejecutar una serie de comandos desde el *powershell* (terminal). Ubicándose en la carpeta en la que se ha creado el proyecto desde el *powershell*, se ejecuta “`yarn install @react-native-firebase/app`”. A continuación, como en este caso se trabaja con una versión *React Native* >= 0.60, sólo hay que ejecutar el comando “`react-native link @react-native-firebase/app`”. Más tarde habrá que generar un archivo desde la plataforma web introduciendo los datos requeridos para ello, descargarlo e incluirlo en el proyecto *VS Code*. Para que este archivo pueda ser utilizado, es necesario agregar estas líneas al archivo *build.gradle: classpath 'com.google.gms:google-services:4.3.3'*. Posteriormente se debe volver a ejecutar el comando “`react-native link`” y ejecutar “`react-native run-android`” y esperar unos segundos para que se vincule la aplicación desde *VS Code* con el proyecto creado desde la plataforma web (*Firebase*).

Para poder utilizar *Firebase Authentication*, primero es necesario instalar la dependencia en el proyecto creado en *VS Code*, para lo cual es necesario ejecutar una serie de comandos desde el *powershell* (terminal). Ubicándose en la carpeta en la que se ha creado el proyecto desde el *powershell*, se ejecuta "`yarn install @react-native-firebase/auth`", a continuación, como en este caso se trabaja con una versión *React Native* >= 0.60, sólo hay que ejecutar el comando "`react-native link @react-native-firebase/auth`". Más tarde se deberá incluir cierta información en el *build.gradle.*

De esta forma ya se pueden utilizar las herramientas para la autenticación y registro mediante *Firebase Authentication*.

Para utilizar correctamente *Firebase Authentication* en el proyecto, además de lo explicado anteriormente, es necesario incluir la instrucción *import auth from '@react-native-firebase/auth';* en cada clase .js en el que se desee utilizar esta funcionalidad. Es necesario utilizar la siguiente notación para realizar las tareas descritas anteriormente:

- ***auth()*** . auth() sirve para poder llamar a métodos acceden a la información almacenada en *Firebase Authentication*.
- ***auth().currentUser*** . Mediante este método se accede a la información sobre el usuario actual.

```
auth().currentUser
```

Imagen 5.2.2. auth()

- ***auth().signInWithEmailAndPassword(email, contraseña)***. Este método permite iniciar sesión con un email y contraseña pasados por parámetros como se indica en la *Imagen 5.2.3*.

```
auth().signInWithEmailAndPassword(email, contrasena)
```

Imagen 5.2.3. método signInWithEmailAndPassword()

- ***auth().signOut()*** . Mediante este método se cierra sesión.

```
auth().signOut()
```

Imagen 5.2.4. método signOut()

- ***auth().createUserWithEmailAndPassword(email, contraseña)*** . Este método crea un usuario con un email y una contraseña que se pasan por parámetro como se indica en la *Imagen 5.2.5*. La información del email y la contraseña se puede extraer del objeto *state* explicado en *3. Sprint 1. Análisis de React Native*.

```
auth().createUserWithEmailAndPassword(this.state.email, this.state.contrasena)
```

Imagen 5.2.5. método createUserWithEmailAndPassword()

En cuanto al diseño gráfico de la aplicación, una vez acabada la parte de autenticación, se ha creado la pantalla 1_Inicio (logging y opción a registrarse) y la de registro de usuarios. La versión final de la pantalla 1_Inicio será la *Imagen 5.2.8.*

```
render(){
  return (
    <ImageBackground style={styles.imageBackground} source={require('../imagenes/backgroundinicio.png')}>
      <View style={{flex: 7}}>
        <View style={styles.mensajeBienvenida}>
        </View>
        <View style={{flex: 1,height: 60,alignItems: "center",justifyContent: "center",marginHorizontal: 30, backgroundColor: 'green'}}>
        </View>
        <View style={{flex: 3, backgroundColor: 'red'}}>
        </View>
        <View style={{flex: 2, alignItems: 'center',backgroundColor: 'green'}}>
        </View>
      </View>
    </ImageBackground>
  )
}
```

Imagen 5.2.6. código pantalla 1_Inicio

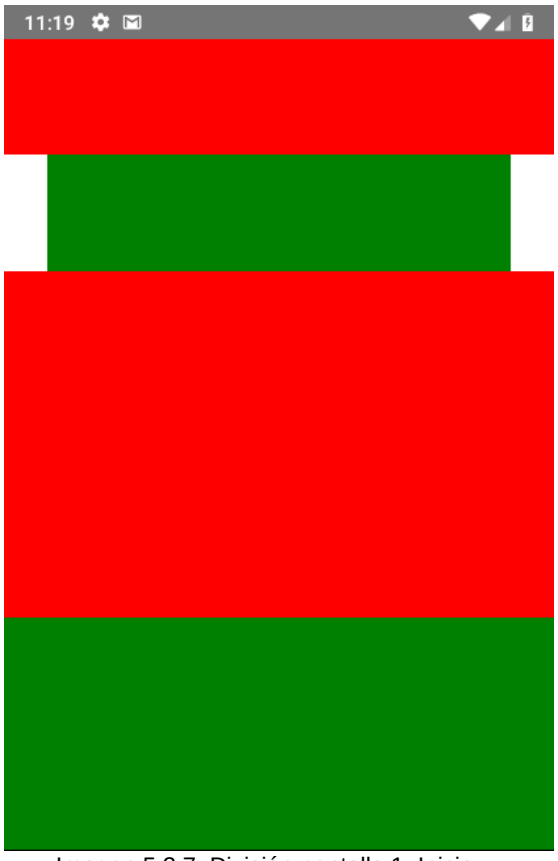


Imagen 5.2.7. División pantalla 1_Inicio

Imagen 5.2.8. Pantalla 1_Inicio definitiva

En lo que refiere al diseño en sí, se está utilizando un diseño basado en el algoritmo *Flexbox* el cual permite tanto dividir la pantalla en áreas de acción, como ordenarlas y justificarlas en tamaño y orientación, adaptándolas al tamaño de la pantalla. Esta forma de trabajo es una de las opciones extraíbles de *JavaScript* y propios de *React Native* para el diseño gráfico. Los botones han sido creados desde el principio partiendo de un componente *TouchableOpacity.*

La pantalla 1_Inicio se ha dividido en 4 áreas de acción en filas. La primera da un mensaje de bienvenida, la segunda está reservada para el espacio en el que se muestran los errores, la tercera está compuesta de un formulario que tiene dos *TextInput* con sus respectivos textos asociados en los que se proporciona la información a pedir y por último, otra área, con dos botones: uno de inicio de sesión, que utiliza *auth* para autenticarse y te envía a la pantalla principal de la aplicación, y otro que lleva a la pantalla 2_Registro tal cual se explica en *4. 2. Prototipado.* La versión final de la pantalla 2_Registro será la *Imagen 5.2.11*.

```
render(){
  const navigation = this.context;
  return (
    <ImageBackground style={styles.imageBackground} source={require('../imagenes/backgroundinicio.png')}>
      <View style={{ flex: 7}}>
        <View style={{flex: 1,height: 72,alignItems: "center",justifyContent: "center",marginHorizontal: 30,backgroundColor: 'red'}}>
        </View>
        <View style={{flex: 4,backgroundColor: 'green'}}>
        </View>
        <View style={{flex: 2, alignItems: 'center',backgroundColor: 'red'}}>
        </View>      You, 2 days ago • echo hasta base de datos (pero con errores)
      </View>
    </ImageBackground>
  )
}
```

Imagen 5.2.9. código pantalla 2_Registro

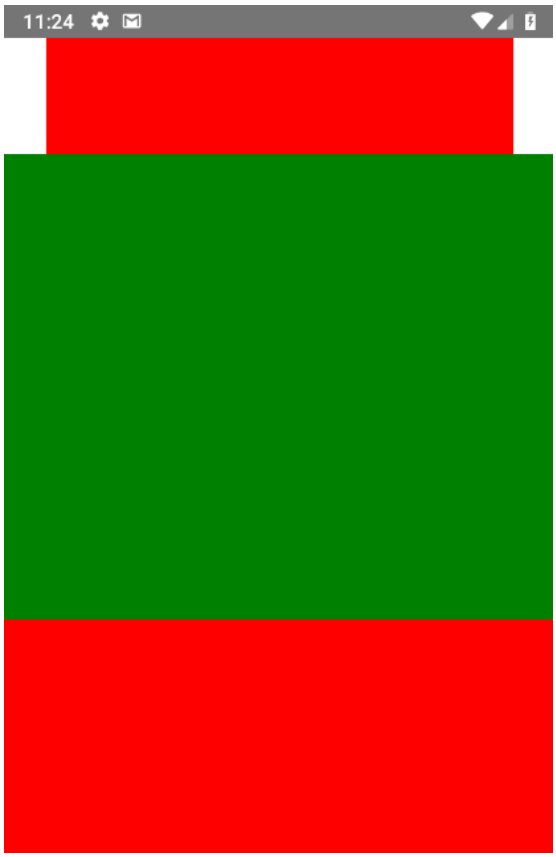


Imagen 5.2.10. División Pantalla 2_Registro

Imagen 5.2.11. Pantalla 2_Registro definitiva

La pantalla 2_Registro está dividida en tres áreas de acción. La primera está reservada para mostrar errores, la segunda contiene un formulario con tres *TextInput* con sus respectivos textos asociados y la tercera contiene dos botones: uno que utiliza *auth* para registrarse y envía un mensaje de alerta con la confirmación del registro, y el segundo botón *Atrás*, que devuelve a la pantalla de Inicio. En ambas pantallas, tanto 1_Inicio como 2_Registro se utiliza una imagen de fondo creada específicamente para este proyecto por una diseñadora gráfica. El modelo fue diseñado inicialmente por Alejandro Ruiz-Olalla de manera esquemática y esta diseñadora lo perfeccionó y digitalizó. A lo largo de los Sprints, al mostrar las pantallas finales, todos los diseños que se muestran, en lo que refiere a iconos y botones, han sido diseñados en papel por el autor del proyecto, y modificados y digitalizados por la diseñadora.

5. 3. MAPAS Y PANTALLA PRINCIPAL

Una vez accedido al corazón de la aplicación, se debe crear una pantalla principal (3_ Pantalla Principal Pinchapp) de tal manera que permita al usuario ir a las diferentes funcionalidades.

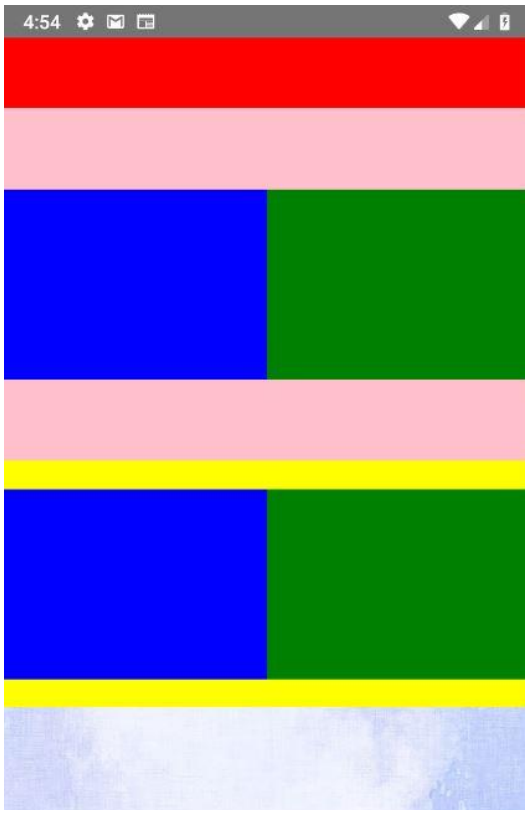


Imagen 5.3.1. División Pantalla 3_Pantalla Principal “Pinchapp”

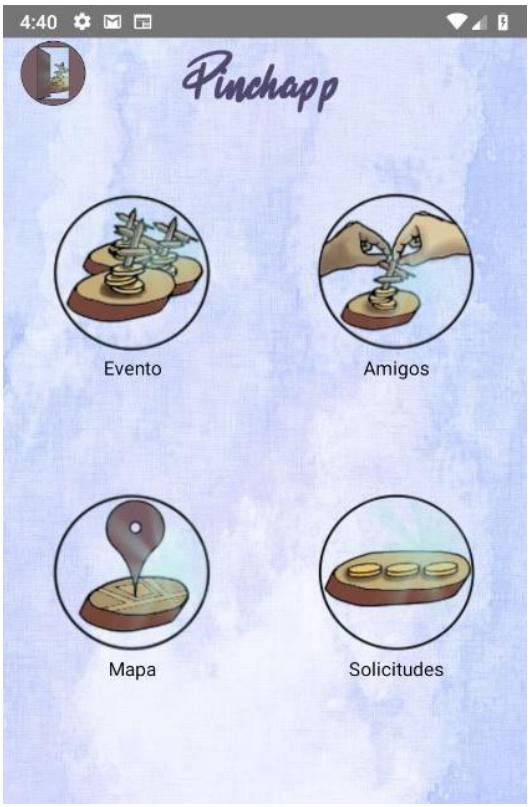


Imagen 5.3.2. Pantalla 3_ Pantalla Principal ”Pinchapp” definitiva

La pantalla 3_Pantalla Principal “Pinchapp”, como se puede observar en la *Imagen 5.3.1,* está dividida en dos áreas de acción. La superior, en la que se crea una cabecera que compartirá elementos con el resto de las pantallas de la aplicación, y cuatro botones. Los botones *Eventos*, *Mapas*, *Amigos* y *Solicitudes* llevaran respectivamente a las pantallas 6_Eventos, 3_Mapas, 5_Amigos y 7_Solicitudes. En la cabecera aparecerá un botón que llama al método *signOut.*

De la misma manera que en el resto de las tecnologías utilizadas en la aplicación, se han tenido que instalar las dependencias de la librería de mapas (*react-native-maps*) antes de poder utilizarla.

Para instalar dicha librería es necesario ubicarse en la carpeta del proyecto mediante el terminal de *powershell* y ejecutar el comando “`yarn add react-native-maps -E`”. Al igual que en las versiones de *Firebase*, al utilizar una versión de *React Native* >= 0.60 para vincular las librerías al proyecto solo es necesario ejecutar el comando “`react-native link react-native-maps`”. A continuación, será necesario utilizar la versión más actualizada de Google Play que se incluye en *build.gradle*. Hubo que realizar el visionado de varios videos explicativos en los que se comprobaba el correcto funcionamiento de esta librería. Se observó que aparecían múltiples opciones de visionado que, por razones de tiempo, no se estudiaron para su aplicación en este trabajo (como, por ejemplo, Street View, 3D View, posibilidad de mostrar o no etiquetas…). Finalmente, en la versión desarrollada, se ha optado por mostrar directamente por pantalla un mapa centrado en La Laurel con el que poder interactuar, para ello se han ajustado los parámetros de latitud y longitud y el *zoom*. Como ya se explicará posteriormente, las opciones de visionado no contempladas en esta primera versión aparecen más adelante como posibles mejoras futuras de la aplicación.

Para utilizar la librería *react-native-maps*, se deberá tener preinstalado en el dispositivo móvil la aplicación Google Maps, ya que esta librería obtiene su información de Google Maps y la muestra por pantalla con las opciones proporcionadas a la misma. La versión final de la pantalla 4_Mapas será la *Imagen 5.3.5*.

```
render(){
  const navigation = this.context;
  return (
    <View style={{flex: 1}}>
      <View style={{flex: 0.4, backgroundColor: 'blue'}}>
      </View>
      <View style={{flex: 4,backgroundColor: 'green'}}>
      </View>
    </View>
  )
}
```

Imagen 5.3.3. código pantalla 4_Mapas

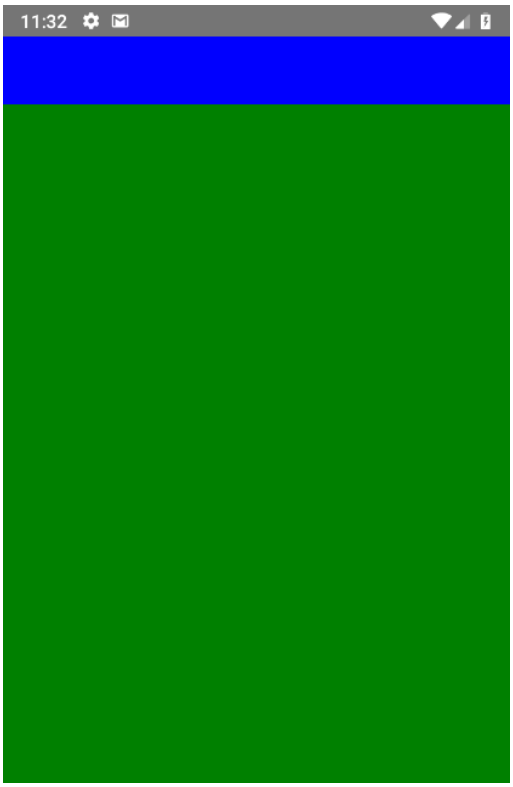


Imagen 5.3.4. Diseño pantalla 4_Mapas

Imagen 5.3.5. Pantalla 4_Mapas definitivo

En esta pantalla, a la que se accede desde 3_Pantalla Principal “Pinchapp” de la aplicación, cuenta con una cabecera similar a la de la pantalla principal, aunque el icono de la izquierda devuelve a 3_Pantalla Principal Pinchapp. En esta clase se puede introducir de manera dinámica tanto la imagen de la izquierda como la acción de hacer clic sobre ella. En la parte inferior se muestra el mapa en sí.

5. 4. SPRINT 3 REVIEW

REVISIÓN DEL SPRINT

Tanto la funcionalidad de Navegación y Registro, como Autenticación discurrieron con total normalidad.

Todos los diseños de las interfaces desarrolladas tanto en este Sprint como en los siguientes (y mostrados en el Anexo 2), incluidos iconos y botones, han sido diseñados en papel por el autor del proyecto y modificados y digitalizados por la diseñadora gráfica.

En el apartado de Mapas, al haber tenido los problemas anteriores con el emulador como se explica en *3. Sprint1.Análisis de React Native*, se tuvo que descargar un PlayStore para incluirlo en *Genymotion*, vincularlo a una cuenta, en este caso la del autor del proyecto, y descargarse desde ese PlayStore una versión actualizada de Google Maps. Se optó por el emulador de *Genymotion*, como se explica en *3. Sprint 1. Análisis de React Native*, el cual no cuenta con un PlayStore por defecto, por lo cual no se pudo acceder a Google Maps. Por consiguiente, se obtuvo el resultado de que la librería *react-native-maps* no mostraba nada por pantalla (además de dar errores de compilación).

A pesar de que las funcionalidades de autenticación, registro y mapas han sido finalizadas con éxito, ciertos aspectos de la aplicación todavía no han podido implementarse. Las acciones de crear un usuario en la base de datos al registrarse, actualizar un campo para saber si un usuario está conectado o no (ya sea para loguearse en la pantalla 1_Inicio, como para cerrar sesión en 3_Pantalla Principal Pinchapp) no se pueden realizar en este Sprint y tendrán que ser implementados a partir del *Sprint 4. Base de datos*.

En este Review, se han realizado una serie de comprobaciones y en caso de estar mal se han corregido.

GITHUB

En cuanto a la visión general del Sprint en el repositorio remoto (*GitHub*), al principio se crea una nueva rama y finalmente, se realiza un *commit* para hacer un *merge* con la rama *develop* y tener la parte funcional del Sprint en la rama principal.

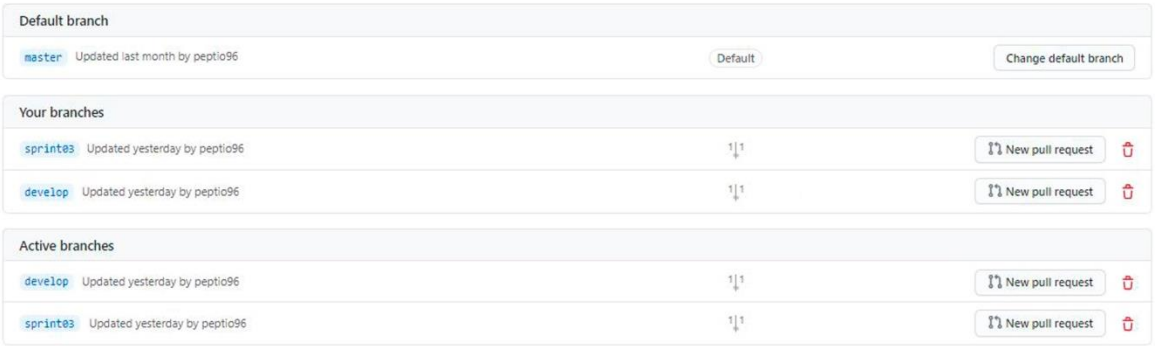


Imagen 5.4.1. Sprint 3 GitHub

6. SPRINT 4. BASE DE DATOS Y PERSISTENCIA

6. 1. BASE DE DATOS

Como se explica en *4. 3. DISEÑO*,la base de datos que se va a crear con *Cloud Firestore* se compone de dos colecciones con una serie de documentos y campos necesarios.

Hemos presentado el diseño de la base de datos, pero para poder implementarla, hay que crear toda la capa de persistencia desde la aplicación creada en *VS Code*. Al contrario que en otras aplicaciones, no se crean los registros desde la propia base de datos, sino que se van creando desde la capa de persistencia de la aplicación.

Para poder trabajar con el flujo de información en la base de datos, se ha implementado el típico s*istema CRUD*. Cada usuario deberá poder realizar una solicitud de amistad de otro usuario (es decir, añadir en la base de datos la correspondiente solicitud de amistad) y aceptar las solicitudes realizadas por otros usuarios en su documento, y leer y escribir en este documento de solicitud. Los usuarios también deberán poder crear eventos y consultar los que estén activos. Si son administradores del evento, podrán incluir más personas, añadir más dinero al evento o eliminar el evento. Los usuarios que no son administradores del evento solo podrán leer el evento del que participan e irse de él. Un evento solo puede leer y escribir su propio documento y los de los usuarios que lo componen, incluido el administrador, para enviarles solicitudes.

Como se ha ido explicando hasta ahora, lo primero que se ha hecho es instalar la dependencia de *Cloud Firestore* mediante el comando "`yarn install @react-native-firebase/firestore`" estando en la *powershell* ubicado en el directorio en el cual se ha creado el proyecto. Posteriormente se ha vinculado al proyecto creado en *VS Code* utilizando el comando "`react-native link @react-native-firebase/firestore`".

Para realizar todas las funcionalidades descritas anteriormente es necesario utilizar un lenguaje propio de *Firebase* que podemos encontrar en *React Native Firebase* (mencionado en *1. 4. Tecnologías*). Mediante la página *React Native Firebase* [16] se ha averiguado cómo usar las tecnologías de *Cloud Firestore*.

Como se ha explicado en *4. 3. Diseño*, *Cloud Firestore* tiene una estructura compuesta de colecciones, que contienen documentos que, a su vez, contienen campos. Para utilizar esta herramienta, y en particular la funcionalidad que proporciona, es necesario utilizar la siguiente notación:

- ***firestore()*** . Sirve para poder llamar a métodos que acceden a la información almacenada en *Cloud Firestore*. Este método es el primero para llamar a los métodos que se especifican posteriormente.
- ***collection()***. Para acceder a una colección, la notación adecuada sería "*firestore().collection('colección a acceder)"* en la que 'colección a acceder' es el nombre de la colección.
- ***doc()*** . Para llamar a un documento dentro de una colección, la notación que se debe utilizar es "f*irestore().collection('colección a llamar').doc('documento a llamar')"* donde 'colección a *llamar*' es la colección en la que está incluido el documento y 'documento a *llamar'* es el identificador del documento al que se quiere acceder.
- ***get()*** . Método que devuelve un objeto tipo JSON con la información de la colección, documento o campo al que se llama. Por ejemplo, en la *Imagen 6.1.1* devuelve el documento "email" de la colección "usuarios".

```
firestore().collection('usuarios').doc(email).get()
```

Imagen 6.1.1. método get()

- ***update()*** . Este método actualiza sobreescribiendo. Puede actualizar tanto documentos como campos dentro de un documento concreto. Por ejemplo, en la *Imagen 6.1.2 se* actualiza el "dinero" de un evento concreto, con identificador "idEvento". El "dinero" se sobreescribe por "dineroPeticion".

```
firestore().collection('eventos').doc(idEvento).update({dinero: documentSnapshot.get('dinero')+parseInt(dineroPeticion)})
```

Imagen 6.1.2. método update()

- ***set()*** . Método que crea o sobreescribe. Puede crear o sobreescribir tanto documentos como campos dentro de un documento concreto. Por ejemplo, en la *Imagen 6.1.3* se actualiza el campo "conectado" de un usuario concreto con identificador "email". El campo "conectado" se sobreescribe por *true*.

```
firestore().collection('usuarios').doc(email).set({conectado: true})
```

Imagen 6.1.3. método set()

- ***where()*** . Este método realiza una consulta al resultado de la colección, documento o campo al que se llama. Esta consulta se realiza con los parámetros que se pasan al método. Por ejemplo, en la *Imagen 6.1.4* se consulta cada documento de la colección "eventos" si tienen un campo "emailAdmin" que sea igual que "emailAdminEvento".

```
firestore().collection('eventos').where('emailAdmin','==',emailAdminEvento)
```

Imagen 6.1.4 método where()

Para crear la capa de persistencia que accede y/o modifica a la base de datos desde el proyecto en *VS Code***,** se crean dos nuevas clases "usuarios" y "eventos", y se almacenan en dos archivos .js en el proyecto de *VS Code* con el mismo nombre respectivo. En estas clases se van a guardar los métodos que acceden y modifican la base de datos. Por ejemplo, los métodos que tiene la clase usuarios van a ser:

- ***registrarse(nombre,email,contraseña)***. Es necesario que tenga los campos nombre "email" y "contraseña" y creará un usuario en *Cloud Firestore* con estos datos y un campo extra llamado "conectado" que automáticamente se inicia a *false*.
- ***logIn(email, contraseña)***. Actualiza el campo "conectado" a *true* del usuario con "email" y "contraseña" iguales a los que se pasan por parámetro.
- ***logOut(email, contraseña)***. Actualiza el campo "conectado" a *false* del usuario con "email" y "contraseña" iguales a los que se pasan por parámetro.
- ***solicitudAmistad(emailEnvia,emailRecibe)***. Se crea un nuevo campo llamado "peticiones" en el usuario con identificador "emailRecibe". Este campo será de tipo *map* y almacena campos *clave-valor* de la siguiente manera: la clave es un número generado de manera aleatoria y el valor es "emailEnvia". Si ya está creado el campo "petición" se añade una nueva solicitud. Si no esta creado, se crea y se añade.
- ***aceptarSolicitudAmistad(emailEnvia,emailRecibe)***. Se elimina el objeto del campo "peticiones" cuyo valor sea "emailRecibe". Se crea un campo "amigos" de tipo *map* en los usuarios con identificador "emailEnvia" y "emailRecibe" que contiene campos *clave-valor*. La clave será un número generado de manera aleatoria y el valor será el "emailRecibe". Si este campo ya estaba creado se añade el objeto. Si el objeto es el último en el campo "peticiones", se eliminará también este campo.
- ***cancelarSolicitudAmistad(emailEnvia,emailRecibe)***. Se elimina el objeto del campo "peticiones" cuyo valor sea "emailRecibe". Si el objeto es el último en el campo "peticiones", se eliminará también este campo.
- ***solicitudesAmistad(email)***. Este método devuelve un objeto tipo JSON con todos los identificadores de los usuarios que han enviado una solicitud de amistad al usuario con identificador "email".

- ***amigos(email)***. Este método devuelve un objeto tipo JSON con todos los identificadores de los usuarios de los que es amigo el usuario con identificador “email”.
- ***solicitudesAmistadEnviadas().*** Este método devuelve un objeto tipo JSON con todos los identificadores de los usuarios a los que el usuario con identificador “email” ha enviado una solicitud de amistad.
- ***solicitudEvento(listaSolicitudes,idEvento,dineroPorPersona)***. Crea o añade el campo “peticionesEventos” con un conjunto de campos en su interior cuyo identificador va a ser generado de manera aleatoria. Va a contener dos campos: “dineroPorPersona” y “nombreEvento”.
- ***aceptarCancelarSolicitudEvento(idEvento,precio,emailAcepta)***. En el usuario con identificador “email” se elimina el objeto del campo “peticionesEventos” cuyo identificador sea el del campo “nombreEvento”.
- ***solicitudesEventos(email)***. Este método devuelve un objeto tipo JSON con todos los identificadores y el dinero por persona de los eventos a los que ha sido invitado el usuario con identificador “email”.
- ***solicitudesEventosEnviadas(email).*** Este método devuelve un objeto tipo JSON con el identificador de los eventos en los que “email” es el propio del campo *emailAdmin* y los “email” de los participantes que están en el campo *noConfirmados* de ese evento.

En lo que refiere a la clase “eventos”, contiene los siguientes métodos:

- ***crearEvento(nombreEvento, emailAdmin,noConfirmados,dineroPorPersona*)**. Se crea un evento en la colección “eventos” con identificador “nombreEvento_emailAdmin”. Tendrá los campos “nombreEvento”, “emailAdmin”, “dinero”, “noConfirmados” y “eventoActivo” con valor *false*.
- ***eventosActivosAdmin(emailAdmin)***. Devuelve un objeto tipo JSON con los eventos en los que el campo “emailAdmin” es “emailAdmin” y el campo “eventoActivo” es *true*.
- ***eventosActivosComen(emailConfirmado)***. Devuelve un objeto tipo JSON con los eventos en los que “emailConfirmado” es el valor de uno de los objetos del campo “confirmados” y “eventoActivo” es *true*.
- ***salirEvento(idEvento, emailComensal)***. Elimina del evento con identificador “idEvento” el objeto con valor “emailComensal” en el campo “confirmados”.
- ***eliminarEvento(idEvento)***. Elimina el evento con identificador “idEvento”.
- ***actualizarEvento(idEvento,emailAcepta*)**. En el evento con identificador “idEvento elimina el objeto con valor “emailAcepta” de “noConfirmados” y lo añade a “confirmados”. Si el campo “noConfirmados” queda vacío, se elimina el campo y se actualiza el valor de “eventoActivo” a *true*.
- ***masBote(idEvento,dineroASumar)***. Se aumenta la cantidad “dineroASumar” al campo “dinero” del evento con identificador “idEvento”.
- ***menosBote(idEvento,dineroARestar)***. Se disminuye la cantidad “dineroASumar” al campo “dinero” del evento con identificador “idEvento”.

6. 2. SPRINT 4 REVIEW

Durante este Sprint, ha habido problemas a la hora de ejecutar el proyecto una vez instalada la dependencia de *Cloud Firestore.* Se observaron fallos en la compilación de esta librería, esto era debido a que se instaló una versión no compatible con las que se estaban utilizando de *Firebase*. Al haberse instalado previamente tanto *@react-native-firebase/app* como *@react-native-firebase/auth* con una última versión que había en el momento y haber tardado unas semanas para instalar *Cloud Firestore*, esta última librería tenía una versión más actualizada no compatible con las anteriores, por lo que hubo que buscar una versión compatible.

Como ya se ha explicado en *5.4. Sprint 3 Review*, era necesaria la creación de la base de datos para poder implementar los métodos que actualizaban el campo *conectado* a *true* o *false.* Dependiendo si se realizaban desde 1_Inicio o 3_Pantalla Principal Pinchapp respectivamente, se han implementado los métodos *logIn*, *logOut*, y los que creaban el usuario para 2_Registro (*registrarse*). En este momento se realiza esta función acabando por tanto las funcionalidades de autenticación y registro.

7. SPRINT 5. AMIGOS Y EVENTOS

En este apartado se explican los aspectos más destacables asociados a la implementación de la funcionalidad referida a la gestión de amigos y eventos.

7. 1. AMIGOS

En particular, en este apartado nos centraremos en los amigos. A continuación, se explica la idea que se ha seguido para implementar cada una de las funcionalidades asociadas con la gestión de amigos, describiendo con más detalle la pantalla 5_Amigos a modo de ejemplo.

- **5_Amigos**. A través de esta pantalla se pueden gestionar los amigos de un usuario dado. En particular, para su diseño, se ha reutilizado la cabecera explicada en *5 .3. Mapas y pantalla principal*. Esta pantalla muestra la información de manera condicional teniendo en cuenta el valor de la variable *hayAmigos*, de forma que, dependiendo de si es *true* o *false*, se ejecutarán las líneas desde la 50 a la 58, o desde la línea 59 a la 65, respectivamente (ver Imagen 7.1.1). La pantalla se divide en dos áreas de acción: la primera incluye la cabecera, y la segunda ocupa el resto de la pantalla (ver *Imagen 7.1.3*). Es decir, en el caso de que el usuario tenga amigos, la segunda área de acción de la pantalla muestra una lista de los elementos creados, cada uno de los cuales están formados por áreas individuales en filas horizontales, compuestas por un icono y un texto que indica el email del amigo. Si, por el contrario, el usuario no tiene amigos registrados, será en la segunda área de acción donde aparece un mensaje que cita “No hay amigos”.
  Para mostrar los elementos creados, se ha utilizado un método al que hemos llamado *amigos*. Este método, creado en la capa de persistencia en *6.1. Base de datos*, devuelve un objeto tipo JSON con todos los identificadores de los usuarios de los que es amigo el usuario con identificador “email”. A partir del archivo que devuelve el método *amigos*, utilizando JSX se crea el componente “todosLosAmigos” como se puede observar en la línea 32 de la *Imagen 7.1.2*. Mediante el método de la línea 32 *Object.values().map()* se recorren todos los amigos del objeto devuelto por *amigos* y se va creando cada elemento de forma dinámica mediante información devuelta por el método anteriormente descrito. La variable *amigo* es la que obtiene el valor de cada amigo cuando se recorre la lista mediante el método *Object.values().map()*. La versión final de la pantalla 5_Amigos es la que se muestra en la *Imagen 7.1.4*.

```
48  <Cabecera/>
49  {this.state.hayAmigos
50    ? <View style={{flex: 4, left: 20}}>
51        <Text>Amigos</Text>
52        <View style={{flex: 1,flexDirection: 'column'}}>
53          <ScrollView style={{marginBottom: 30}}>
54            {listaAmigos}
55          </ScrollView>
56          <View style={{flex: 2}}/>
57        </View>
58      </View>
59    : <View style={{flex: 4, left: 20}}>
60        <View style={{flexDirection: 'row'}}>
61          <View style={{flex: 2, left: 20}}>
62            <Text>No hay amigos</Text>
63          </View>
64        </View>
65      </View>
66  }
```

Imagen 7.1.1 código pantalla 5_Amigos (I)

```
32    const todosLosAmigos = Object.values(amigos).map((amigo) => {
33      return(
34        <View key={amigo} style={{flex: 1, height: 50, flexDirection: 'row'}}>
35          <View style={{width:50,height: 50}}>
36            <Image source={require('../imagenes/logo_persona.png')}></Image>
37          </View>
38          <View style={{height: 50,width: '80%', flexDirection: 'column'}}>
39            <Text >{amigo}</Text>
40          </View>
41        </View>
42      )
43    })
```

Imagen 7.1.2 código pantalla 5_Amigos (II)

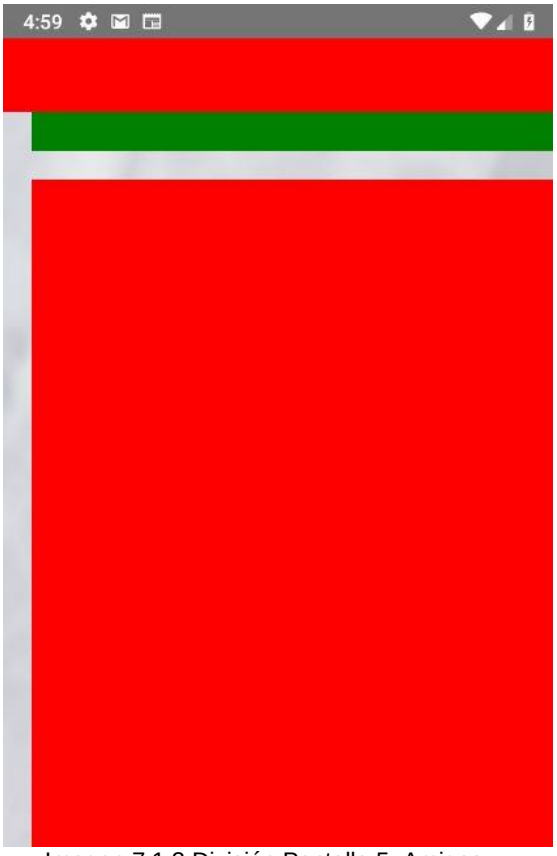


Imagen 7.1.3 División Pantalla 5_Amigos

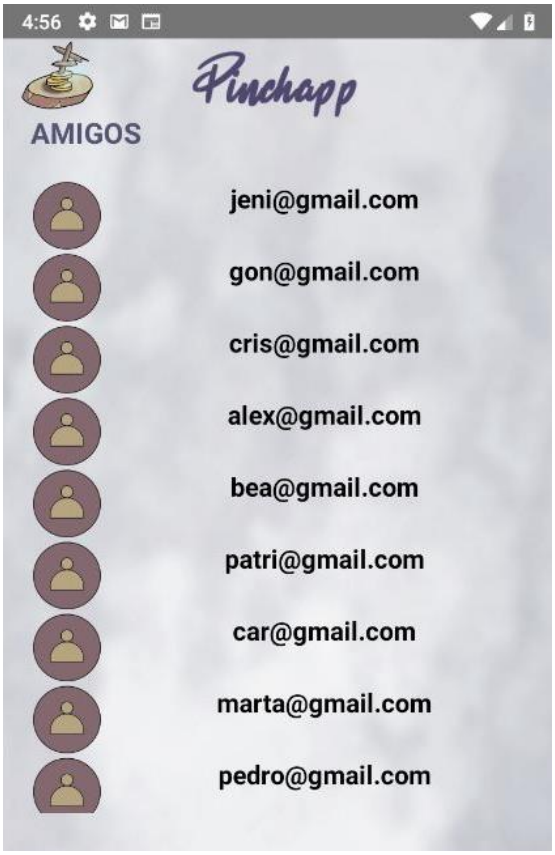


Imagen 7.1.4 Pantalla 5_Amigos definitiva

- **7_Solicitudes**. Para el desarrollo de esta pantalla, se ha utilizado el método *solicitudAmistad*, teniendo en cuenta los posibles errores que pueda producir a la hora de introducir la información. Se ha tenido en cuenta también que no se debe poder enviar una solicitud a un usuario dos veces a un usuario que ya sea amigo ni a uno mismo. Se reutilizan la cabecera explicada en *5 .3. Mapas y pantalla principal*. Se divide la pantalla en dos áreas de acción principales, la cabecera y el resto del cuerpo. Este último campo se divide a su vez en tres columnas, las dos de los extremos sirven para colocar los botones de flecha para desplazarse a las pantallas "7_1_Solicitudes Amistad" y "7_2_Solicitudes Eventos", quedando en la columna central otras dos áreas, una en el encabezado superior reservada para errores.

- **7_1_Solicitudes Amistad.** En esta pantalla se van a representar las listas de las solicitudes de amistad recibidas y enviadas. Para representar las dos listas se van a utilizar las mismas técnicas que en la pantalla 5_Amigos, pero en la lista de solicitudes recibidas, cada una de ellas contará con dos botones: *Aceptar* y *Cancelar*. Al aceptar y cancelar se utilizan los métodos correspondientes creados previamente en la capa de persistencia. Para representar la lista de solicitudes enviadas se utiliza el método *solicitudesAmistadEnviadas*.

7. 2. EVENTOS

En el momento de creación de un evento, debemos pasar la información introducida para la creación del evento desde la pantalla "6_1_Crear Evento" hasta "6_1_1_Añadir comensales", que es donde realmente se completa la creación del evento. Por ello, sería necesario transferir la información introducida en los dos *inputs* desde "6_1_Crear Evento" a "6_1_1_Añadir comensales". Para pasar la información de una pantalla a otra, se optó por añadir un nuevo parámetro en el método "*navigation.navigate('nombre de la pantalla a la que navegar',{})",* a través del cual pasar esa información a la pantalla de destino.

- **6_Evento**. Se sigue la misma estructura gráfica que en "3_Pantalla Principal Pinchapp".
- **6_1_Crear Evento.** Como ya se ha explicado en *5. 1. Navegación*, para pasar de una pantalla a otra, se utiliza "*navigation.navigate('nombre de la pantalla a la que navegar')"*. Para economizar la información en la base de datos se encontró que al escribir "*navigation.navigate('nombre de la pantalla a la que navegar',{})"*, dentro de los corchetes se puede introducir un objeto .json de tal forma que de la pantalla a la que vas se puede acceder mediante "*this.props.params.'nombre del objeto .json'* ". En esta pantalla se pasa la información referida tanto al nombre del evento como del dinero que cobrar a cada comensal. Se reutilizan componentes gráficos para la creación de esta pantalla.
- **6_1_1_Añadir comensales.** Se utilizan técnicas anteriormente descritas para mostrar los amigos del usuario con un botón para enviar solicitudes de eventos. Aparecen dos botones más *Crear evento* y *Cancelar*, *Cancelar* envía al usuario a la pantalla "6_Evento". *Crear evento* llama al método (*crearEvento*) creado en la capa de persistencia que accede a la base de datos y crea un nuevo documento en la colección *eventos* con los datos proporcionados en la pantalla "6_1_Crear Evento".
  Al acabar de crear el evento se envían solicitudes de evento a los participantes en los que se incluye el dinero a cobrar por persona. En este momento se deja preparado el código de tal manera que en el *Sprint 6. Pagos y transferencias* se pueda proceder a la introducción de los métodos que permitirían su finalización, aunque en este momento no se estén realizando. Al administrador se le suma automáticamente la cantidad especificada y se muestra un *alert* en el que se informa de que se ha creado el evento y que se ha cobrado el dinero pertinente. Se reutilizan componentes gráficos para la creación de esta pantalla. La versión final de la pantalla 6_1_1_Añadir comensales será la *Imagen 7.2.2*.

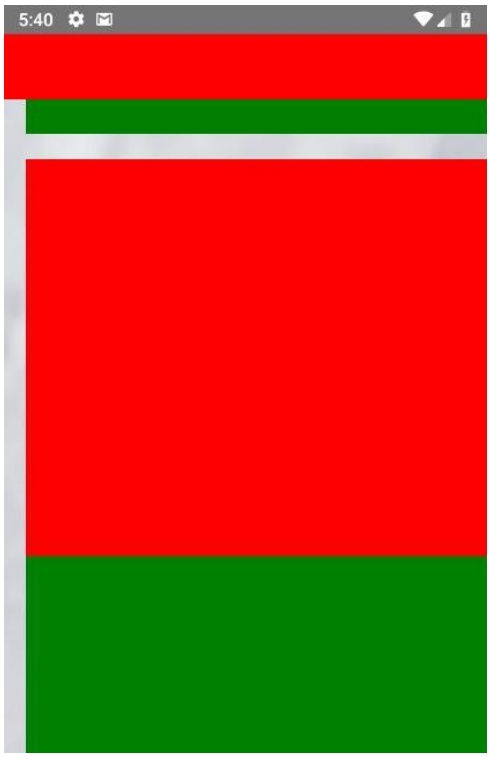


Imagen 7.2.1 División Pantalla 6_1_1_Añadir comensales

Imagen7.2.2 Pantalla 6_1_1_Añadir comensales definitiva

- **6_2_Eventos Activos.** Utilizando las mismas técnicas que en pantallas anteriores se accede a la base de datos para mostrar dos listas utilizando los métodos *eventosActivosAdmin* y *eventosActivosComen.* Al hacer clic en cualquiera de los elementos de las dos listas se accede a las pantallas "6_2_1_Evento Administrador" y "6_2_2_Evento Participante". Se reutilizan componentes gráficos para la creación de esta pantalla.
- **6_2_1_Evento Administrador.** Las opciones que se muestran en esta pantalla van enviando al usuario a diferentes pantallas excepto el botón *Acabar evento* que llama al método *eliminarEvento* y envía al usuario a "6_2_Eventos Activos".
- **6_2_1_1_Aumentar Bote.** Al introducir el dinero y hacer clic en *Aumentar*, se utilizan los métodos *masBote* y *solicitudEvento* para aumentar el "bote" y enviar una solicitud de evento con el importe a aumentar a cada participante del evento. Una vez aumentado el bote y enviadas las solicitudes, se lleva al usuario a la pantalla "6_2_Eventos Activos
- **6_2_1_2_Pagar.** En esta pantalla se realizarán las funciones explicadas en *8. Sprint 6. Pagos y transferencias*.
- **6_2_2_Evento Participante.** Al hacer clic en el botón *Salir de evento* se llama al método *salirEvento,* acción que llevará al usuario a la pantalla "6_2_Eventos Activos".
- **7_2_Solicitudes Eventos.** Para la creación de esta pantalla se utilizan los métodos *solicitudesEvento, solicitudesEventoEnviadas*, *actualizarEvento*, y *aceptarSolicitudEvento* y *cancelarSolicitudEvento* que se asocian a los botones con el mismo nombre respectivo. En cuanto a la parte gráfica de la pantalla, sigue el mismo esquema general que "7_Solicitudes" en cuanto a la división en columnas con un botón izquierdo. Se utiliza el esquema de "5_Amigos" para el diseño de las solicitudes pendientes a eventos y las solicitudes enviadas. En la parte de solicitudes enviadas, por cada evento al que se ha enviado una solicitud aparece el icono evento, el nombre del evento y una lista con *scrollbar* de los nombres de los participantes que se encuentran en el campo *noConfirmados* de ese evento.

7. 3. SPRINT 5 REVIEW

REVISIÓN DEL SPRINT

Dado que *React Native* ofrece la posibilidad de reutilizar componentes y que muchas de las pantallas creadas en este Sprint son similares gráficamente, el tiempo estimado de duración se ha visto reducido.

GITHUB

En cuanto a la visión general del Sprint en el repositorio remoto (*GitHub*), se crea al principio una nueva rama y finalmente se realiza un *commit* para hacer un *merge* con la rama *develop* y tener la parte funcional del Sprint en la rama principal. Este Sprint se realizó en un tiempo menor al estimado puesto que, aunque al principio fue farragoso, una vez aprendido a realizar los componentes necesarios para la creación de una pantalla, *React Native*, al permitir la reutilización de los mismos hace que sea más rápido.

Imagen 7.3.1. Sprint 5 GitHub

8. SPRINT 6. PAGOS Y TRANSFERENCIAS

8.1. MÉTODOS DE PAGO

La aplicación en un principio debería gestionar transferencias de dinero en varios contextos: para realizar los pagos en los establecimientos y transferencias a “eventos” y “usuarios”. Para implementar estas funcionalidades, se han tenido que estudiar y valorar diferentes métodos de pago, no obstante, tras muchas horas y esfuerzo invertido, no se ha podido implementar ninguno de manera correcta, ya que todos eran de pago o, en el caso de *Docker*, no estaba adaptado para utilizarse en el desarrollo móvil.

Por esta razón en este Sprint nos centraremos en explicar nuestra experiencia y lecciones aprendidas en lo que refiere a dichos métodos de pago analizados, de tal manera que, si en un futuro se dispusiera la comercialización de la app, ya contemos con ese trabajo realizado.

El primer método de pago estudiado fue *Bizum*, y fue el sistema que Javier nos sugirió utilizar en un principio. Por este motivo, se decidió enviar un mensaje a la compañía Sociedad de Procedimientos de Pago, S.L (empresa que controla *Bizum*). Desde la empresa se recibió un mensaje con un enlace que llevaba a una página de información que explicaba cómo utilizar *Bizum* dentro de un TPV Virtual, razón por la que se buscó un TPV Virtual para llevar a cabo este método. Se llamó a varias entidades bancarias para adquirir esta tecnología y resultó que era una herramienta de pago para empresas, razón por la que se buscó un TPV Virtual independiente. El tutor de empresa proporcionó una página web en la que se podía acceder a esta herramienta, pero eran necesarios conocimientos de *Docker* y *PHP*.

Al estudiar cómo encontrar un TPV Virtual se encontró uno escrito en *PHP* desde un repositorio público de *GitHub*. Para utilizarlo, Joan Roca propuso utilizar como contenedor del TPV Virtual, *Docker* y almacenarlo en un servidor gratuito de Amazon. A partir de este momento, se comenzó a estudiar *Docker* de forma local. Para ello se hicieron varias pruebas hasta que se consiguió instalarlo en el ordenador y correrlo por primera vez utilizando una serie de comandos (`docker pull php:7.0-apache`, `docker run-p 9090:80-v'$PW':/var/www/html –name=me-servidor…`) desde el *powershell*. A raíz de lo aprendido, se consiguió mostrar la información del *PHP* en una página web. Sin embargo, no se consiguió que apareciera ningún tipo de información desde la aplicación, ya que la información estaba preparada para utilizarse en desarrollo web. Al intentar trasladarlo al desarrollo móvil, se requerían conocimientos más avanzados para poder aprenderlo e incluirlo en este TFG.

El siguiente método de pago en el que se pensó fue buscar una librería propia de *React Native* para realizar pagos sin necesidad de utilizar un TPV Virtual. Para ello se encontró la librería *react-native-payments* [29] desde la cual se podían realizar pagos utilizando *Google Pay* y *Apple Pay*. Tras varios intentos fallidos para instalar esta dependencia se buscó otra librería, en particular, se probaron varias hasta que se llegó a *tipsi-stripe*[28].

La librería *tipsi-stripe* fue la única que se consiguió instalar correctamente como dependencia del proyecto. A pesar de que también había que pagar para utilizar esta tecnología de pago, la librería se podía instalar antes de realizar ninguna transferencia, como versión de prueba. Se decidió, por lo tanto, trabajar la funcionalidad de pago en establecimientos desde la versión de prueba.

Stripe[34] es una compañía tecnológica gracias a la cual se pueden realizar pagos por internet. El propio software de la compañía proporciona herramientas para la seguridad de las cuentas. Para poder utilizar el software, hay que crearse una cuenta en la página web, y al activarla se proporciona una clave necesaria para poder utilizar la librería *tipsi-stripe* (de la misma compañía). Para la activación de esta cuenta es necesario incluir una cuenta bancaria para que se realice el pago por el uso de la librería y proporcionar una serie de información como el nombre de la empresa, el CIF etc.

8.2. PAGOS Y TRANSFERENCIAS

En particular para poder utilizar *tipsi-stripe* para realizar los pagos en establecimientos, hay que realizar una serie de pasos previos.

Como se puede ver en la *Imagen 8.2.1*, para poder utilizar *tipsi-stripe* es necesario proporcionar una serie de parámetros para verificar que se tiene cuenta de *Stripe* y que está activa, como *publishableKey* (en la línea 15) y se le puede añadir en el parámetro *androidPayMode*, *test* o *production* (en la línea 16). La propia librería de *Stripe* proporciona un entorno controlado de pruebas para verificar el correcto funcionamiento de la dependencia. Existen también parámetros opcionales como *merchantId* (en la línea 17), es decir el nombre del comerciante que se pueden incluir como información extra. Al ser imposible la activación de la cuenta en *Stripe* por no ser una empresa, el parámetro *publishableKey* tiene un valor no válido y en *androidPayMode* tiene incluido *test*.

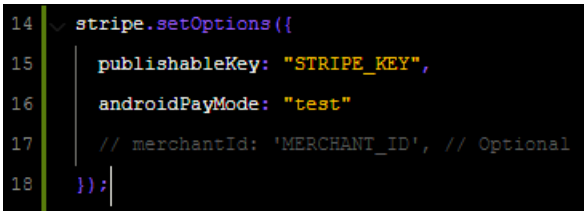
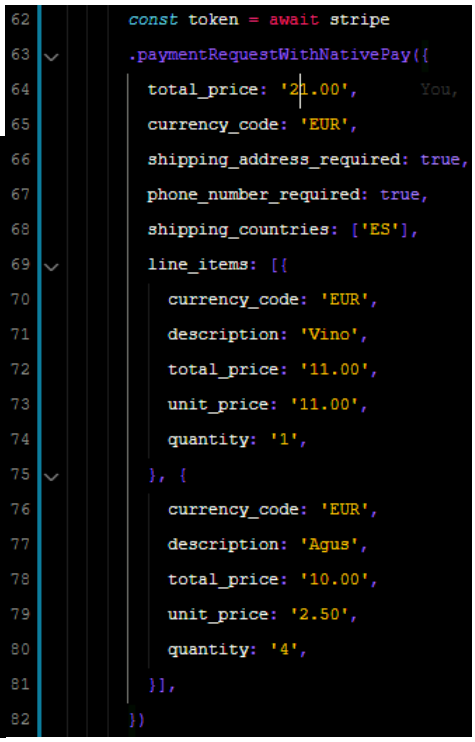


Imagen 8.2.1. código pantalla 6_2_1_2_Pagar (I)

Como se puede observar en la *Imagen 8.2.2* en la línea 140, hay tres puntos que significa que se añade a lo que había anteriormente, *testID*, que es una comprobación en el sistema Android de que *publishableKey* es correcta. En la línea 139 se llama en la opción *onPress* de *TouchableOpacity handleAndroidPayPress* que crea un *token* con: el precio total a cobrar, de qué *ítems* está compuesto el precio, la moneda utilizada en la transacción, si se paga por teléfono móvil o por número de cuenta de Android Pay... El *token* que se puede ver en la *Imagen 8.2.3* ha sido creado de manera manual para poder ver un ejemplo de lo que se vería cuando se active la cuenta *Stripe*, pero este sería generado por Android Pay al pasar el móvil por el datáfono. Entre las líneas 148 y 152 se mostraría el *token* que devuelve la acción de pagar con Android Pay.

```
<View style={styles.pagar}>
  <TouchableOpacity style={styles.boton}
    onPress={this.handleAndroidPayPress}
    {...testID('androidPayButton')}>
    <Text style={{ color: "#FFF", fontWeight: "500" }}>
      Pagar con Android Pay
    </Text>
  </TouchableOpacity>
  <View
    style={styles.token}
    {...testID('androidPayToken')}>
    {token &&
      <Text style={styles.instruction}>
        Token: {token.tokenId}
      </Text>
    }
  </View>
</View>
```

Imagen 8.2.2. código pantalla 6_2_1_2_Pagar (II)



```
const token = await stripe
.paymentRequestWithNativePay({
  total_price: '21.00',
  currency_code: 'EUR',
  shipping_address_required: true,
  phone_number_required: true,
  shipping_countries: ['ES'],
  line_items: [{
    currency_code: 'EUR',
    description: 'Vino',
    total_price: '11.00',
    unit_price: '11.00',
    quantity: '1',
  }, {
    currency_code: 'EUR',
    description: 'Agus',
    total_price: '10.00',
    unit_price: '2.50',
    quantity: '4',
  }],
})
```

Imagen 8.2.3. código token

Para poder realizar los pagos en establecimientos, habría que activar la cuenta en *Stripe* de tal manera que se proporcionara la clave que introducir en el valor *publishableKey*. Después de introducir este dato, se realizarían las pruebas de comprobación de funcionamiento de este método (*androidPayMode*: *test*). Una vez probado el funcionamiento, cuando se observe que funciona de manera correcta, se cambiará *androidPayMode* de *test* a *production*. De esta manera, ya estará dispuesto el código para proceder a los pagos en establecimientos.

Al no poder realizar las pruebas descritas anteriormente para los pagos en establecimientos, ha sido imposible el estudio tanto de las transferencias de usuarios a eventos y viceversa, como del aumento del dinero en el bote. Por este motivo, dichas funcionalidades no han podido ser completadas.

8.3. SPRINT 6 REVIEW

REVISIÓN DE FUNCIONALIDADES

Teniendo en cuenta lo anterior, en este Sprint, las funcionalidades de aumento de dinero (R11), transferencias de usuarios a eventos (R5 y R7) y de eventos a usuarios (R8 y R9) se dejan incompletas, de tal manera que una vez se pudiera activar la cuenta en *Stripe*, hubiera que introducir los métodos para realizar este tipo de transferencias. Desde *6. Sprint 4. Base de datos* cada vez que se necesita cobrar una cantidad se cuenta con un campo *dinero* que proporciona el valor de la cantidad a pagar.

La funcionalidad de cobro en establecimientos (R10) se ha dejado también preparado para utilizar, y únicamente habría que cambiar el valor de *publishableKey* por la administrada desde la página *Stripe* y el token del que se habla en este Sprint. Una vez realizados estos cambios la funcionalidad estaría finalizada.

En conjunto, el estudio de las funcionalidades, la búsqueda de librerías y la implementación de *tipsi-stripe*, han requerido de más tiempo del previsto en este Sprint.

GITHUB

En cuanto a la visión general del Sprint en el repositorio remoto (*GitHub*), se crea al principio una nueva rama y se realiza un *commit* para hacer un *merge* con la rama *develop* y tener la parte funcional del Sprint en la rama principal.

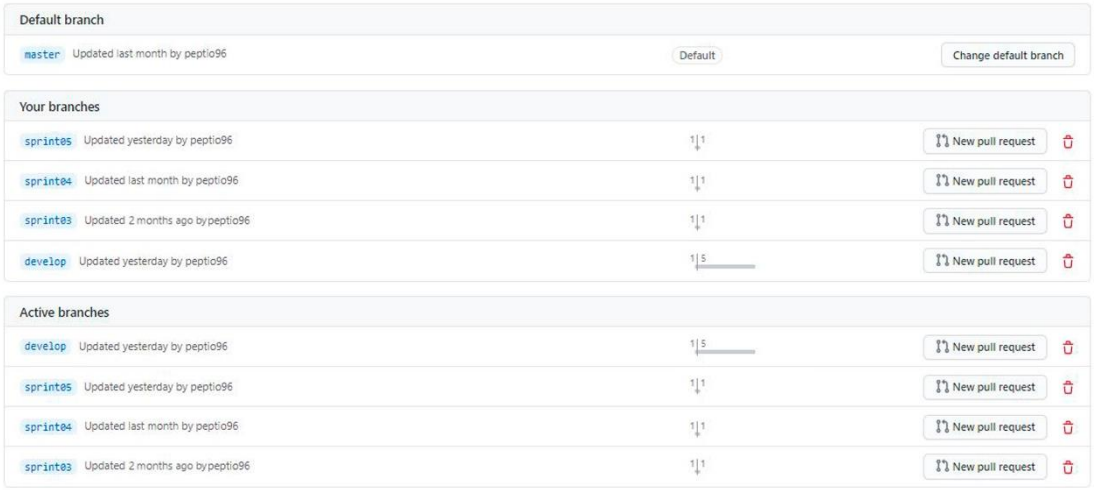


Imagen 8.3.1. Sprint 6 GitHub

9. SEGUIMIENTO Y CONTROL

En cuanto al tiempo estimado de realización del TFG, los problemas sucedidos a raíz de la pandemia generaron un desfase con relación al tiempo real que finalmente se ha dedicado a desarrollar este proyecto. A continuación, se muestran dos tablas que reflejan la distribución real en semanas de los diferentes Sprint en contraposición a la *Imagen 2.2.3* e *Imagen 2.2.4* que explicaban la organización estimada.

Aunque se comenzó el TFG en el momento que se estimó, al final las asignaturas de la universidad (unido con el confinamiento por la pandemia), provocaron que, entre los meses de abril y mayo, no se pudiera trabajar en el proyecto. Se comenzó de nuevo el desarrollo del proyecto una vez acabada la asignatura de Seguridad. Durante estos primeros meses de febrero y marzo, principalmente se analizaron las aplicaciones y el lenguaje. También se comenzaron a estudiar las tecnologías de pago y es en este momento en el que se observó que la mayoría no eran gratuitas y se dejó el estudio de las mismas para más adelante.

Nº Actividad	Horas al día de media	Inicio	Final	Días	Horas	semana 3-9 febrero	semana 10-16 febrero	semana 17-23 febrero	semana 24feb-1 marzo	semana 2-8 marzo	semana 9-15 marzo	semana 16-22 marzo	semana 23-29 marzo	semana 30marzo-5abril	semana 6-12 abril	semana 13-19 abril	semana 20-26 abril	semana 27abril-3mayo	semana 4-10mayo	semana 11-17mayo
Sprint 0 Análisis de herramientas requisitos finales y planificación	5,0	03/02/2020	14/02/2020	10	50															
Sprint 1 Análisis React Native	5,0	17/02/2020	27/03/2020	20	100															
Sprint 2 Control de versiones y diseño	1,5	01/06/2020	19/06/2020	15	23															
Sprint 3 Autenticación y mapas	1,0	22/06/2020	03/07/2020	10	10															
Sprint 4 Base de datos	3,0	06/07/2020	17/07/2020	10	30															
Sprint 5 Amigos y Eventos	2,0	20/07/2020	31/07/2020	10	20															
Sprint 6 Pagos y transferencias	3,0	24/02/2020	21/08/2020	25	75															
Memoria	2,5	24/08/2020	28/08/2020	10	25															
TOTAL				110	333															

Imagen 9.1 Tabla final de organización de tiempo de los Sprints (I).

Nº Actividad	Horas al día de media	Inicio	Final	Días	Horas	semana 18-24mayo	semana 25-31mayo	semana 1-7junio	semana 8-14 junio	semana 15-21 junio	semana 22-28 junio	semana 29junio-5 julio	semana 6-12 julio	semana 13-19 julio	semana 20-26 julio	semana 27julio-2 agosto	semana 3-9 agosto	semana 10-16 agosto	semana 17-23 agosto	semana 24-30 agosto	semana 31 ago-6 septiembre
Sprint 0 Análisis de herramientas requisitos finales y planificación	5,0	03/02/2020	14/02/2020	10	50																
Sprint 1 Análisis React Native	5,0	17/02/2020	27/03/2020	20	100																
Sprint 2 Control de versiones y diseño	1,5	01/06/2020	19/06/2020	15	23																
Sprint 3 Autenticación y mapas	1,0	22/06/2020	03/07/2020	10	10																
Sprint 4 Base de datos	3,0	06/07/2020	17/07/2020	10	30																
Sprint 5 Amigos y Eventos	2,0	20/07/2020	31/07/2020	10	20																
Sprint 6 Pagos y transferencias	3,0	24/02/2020	21/08/2020	25	75																
Memoria	2,5	24/08/2020	28/08/2020	10	25																
TOTAL				110	333																

Imagen 9.2 Tabla final de organización de tiempo de los Sprints (II).

	Horas estimadas	Horas reales	% Desfase
Sprint 0 Análisis de herramientas requisitos finales y planificación	50	50	0%
Sprint 1 Análisis React Native	100	100	0%
Sprint 2 Control de versiones y diseño	30	22,5	25%
Sprint 3 Autenticación y mapas	15	10	33%
Sprint 4 Base de datos	30	30	0%
Sprint 5 Amigos y Eventos	30	20	33%
Sprint 6 Pagos y transferencias	30	75	-150%
Memoria	15	25	-67%
TOTAL	300	332,5	-11%

Imagen 9.3. Resumen de horas

Como se puede ver en la tabla anterior, el desfase en horas es especialmente significativo en el Sprint 6, ya que se ha necesitado invertir mucho más tiempo del esperado, por la dificultad de las funcionalidades a desarrollar en los mismos. De hecho, no se ha conseguido llegar al resultado esperado ya que se necesitaba disponer de un CIF de empresa y realizar una transferencia a la compañía *Stripe* por lo que no se pudo llevar completamente a cabo. En cuanto a Memoria, a pesar de ser un apartado en principio fácil de realizar, se tardó mucho más tiempo puesto que había muchos fallos que, gracias a la tutora se pudieron resolver, aunque tardando algo más. Respecto al Sprint 3. Autenticación y mapas, se estimó una cantidad de tiempo mayor, puesto que al tener que utilizar un lenguaje nuevo, no se sabía la dificultad del mismo. Al final fue más asequible de lo esperado, gracias a *Firebase Authentication*.

10. CONCLUSIONES

VISIÓN A FUTURO

Dentro del estudio de la aplicación, hay opciones y posibilidades que han quedado fuera del proyecto por razones económicas o de carga de trabajo. En este apartado se explican posibles mejoras que se podrían hacer a la aplicación “Pinchapp”, siendo las mismas una proyección a futuro en caso de que acabe siendo comercializada.

- **Pagos**. En primer lugar, y como necesidad imprescindible en el caso de que Pinchapp pase a comercializarse, tendría que utilizarse una forma de pago estable de cualquiera de las maneras ya explicadas en *8. Sprint 6. Pagos y transferencias.*
- **Perfil**. Una posible mejora sería considerar más información del usuario en el perfil, como, por ejemplo, una fotografía o imagen, su edad, gustos gastronómicos etc., ya que en la actualidad solamente se registra el email, la contraseña y los amigos. Otra mejora en esta línea sería además permitir editar el perfil dando la opción de cambiar cualquier aspecto de esa información.
- **Mapas**. Estudiar las opciones de mapas que ofrece Google Maps e implantar las importantes: geolocalización, visión o no de los bares y establecimientos, opciones de Street View etc.
- **Amigos**. Habilitar la eliminación de amigos del perfil del usuario. Poder crear grupos de amigos para facilitar la creación de eventos con usuarios fijos.
- **Eventos**. Dar la opción de visualizar eventos que ya no estén activos y permitir la revisión del dinero gastado por cada uno de los comensales de manera gráfica.

ASPECTOS TECNOLÓGICOS

El lenguaje *React Native* comenzó siendo complejo de utilizar por su novedad y falta de documentación. La curva de aprendizaje en un principio es más lenta, pero se agiliza a lo largo del tiempo de manera exponencial. Una vez entendido el lenguaje, las facilidades para su trabajo con relación a las aplicaciones móviles supera a otros como *Android Studio*, ya que permite economizar código y reutilizar componentes. Además, gracias a las plataformas como *Firebase*, se consigue un *backend* muy rápido y eficiente.

Con relación a las tecnologías de realización de pagos, se ha aprendido que es más complejo de lo que aparentaba en un principio y considero que en *React Native* las transferencias bancarias son un aspecto a mejorar. Se sobreentiende que al ser un lenguaje nuevo y en continuo desarrollo, las librerías que proporcionen métodos de pago irán creándose y creciendo de tal manera que en el futuro próximo sea más fácil implementarlas.

Gracias a la asignatura DIU (Diseño de Interfaces para Usuario), este trabajo ha sido más fácil de desarrollar en el tema gráfico. La planificación de posibilidades, flujos de navegación y funcionalidades es vital para un buen progreso gráfico en aplicaciones.

CUMPLIMIENTO DE REQUISITOS

Como se puede observar a lo largo de los Sprints, se han cumplido los requisitos funcionales de autenticación y registro (R1), amistad entre usuarios (R2), acceso a mapas (R3), creación de eventos (R4) y consulta de eventos (R6). En incorporación al evento (R7), abandono del evento (R8), eliminación del evento (R9) y aumento de dinero (R11) las funcionalidades han sido cumplimentadas a excepción de los apartados de pagos y transferencias entre los usuarios y eventos y eventos y usuarios, tal y como se ha explicado en el Sprint 6. Pagos y transferencias. En cuanto a cobro de eventos (R5), y pagos (R10) no ha sido posible su ejecución por las razones explicadas en Sprint 6. Pagos y transferencias.

En cuanto a los requisitos no funcionales, el requisito de compatibilidad, según las pruebas realizadas en distintos dispositivos móviles, como teléfonos y tablets, se cumple. Si se prueba en versiones anteriores de Android, la aplicación “Pinchapp” puede no permitir la instalación. En cuanto al requisito de legislación, Firebase posee certificaciones según los marcos de trabajo de la Unión Europea como se puede observar en la *Imagen 10.1.*

Nombre del servicio ▾	ISO 27001	ISO 27017	ISO 27018	SOC 1	SOC 2	SOC 3
Cloud Firestore	✓	✓	✓	✓	✓	✓
Cloud Functions para Firebase	✓	✓	✓	✓	✓	✓
Cloud Storage para Firebase	✓	✓	✓	✓	✓	✓
Firebase A/B Testing	✓			✓	✓	✓
Firebase Authentication	✓	✓	✓	✓	✓	✓

Imagen10.1 Cumplimiento de ISO y SOC

Gracias a la realización de este TFG me he dado cuenta de las dificultades que plantea la creación de una aplicación si se hace de manera individual, me ha ayudado a comprender la necesidad del trabajo en grupo y la razón por la que Informática es un área en la que prima el compañerismo. La organización es muy importante, así como seguir un orden y sobretodo tener a alguien que vaya controlando los tiempos y el ritmo de trabajo, que en este caso ha sido mi tutora. En el futuro me gustaría continuar con la realización de las funcionalidades que han quedado sin implementar o a medias y con las mejoras que se han propuesto en *Visión a futuro*.

AGRADECIMIENTOS

Me gustaría agradecer a mi tutora Beatriz Pérez Valle por su constancia y el esfuerzo que ha realizado en la multitud de correcciones que me ha ofrecido, sin ella no habría sido posible la finalización de este trabajo ni este resultado final del que estoy orgulloso. Tanto Javier Virto como Joan Roca también han participado de este proyecto, sobretodo al principio del mismo y a la hora de entender cómo iba a ser Pinchapp, por lo que también les agradezco su disposición. En último lugar, me gustaría agradecer a Cristina Ruiz-Olalla, que me ha proporcionado las herramientas gráficas y me he dado cuenta de que además de ser una buena hermana es también una excelente profesional.

11. BIBLIOGRAFÍA

[1]Square disponible en: https://squareup.com/us/en?country_redirection=true . Accedido en febrero de 2020.

[2]Momopocket disponible en: https://www.momopocket.com/ . Accedido en febrero de 2020.

[3]*Google Pay* disponible en:

https://pay.google.com/about/?gclid=Cj0KCQiAkePyBRCEARIsAMy5Scu7LGbkSyVv8_kJudn_mHKl355AZfjWhqxI9kDP-zlPW0IWF895QR8aAuIeEALw_wcB . Accedido en febrero de 2020.

[4]Twyp disponible en: https://www.twyp.com/ . Accedido en febrero de 2020.

[5]Verse disponible en: https://verse.me/es/ . Accedido en febrero de 2020.

[6]Splitwise disponible en: https://www.splitwise.com/ . Accedido en febrero de 2020.

[7] *Virtual Box* disponible en: https://www.virtualbox.org/. Accedido en febrero de 2020.

[8] *React* disponible en: https://es.reactjs.org/ . Accedido en febrero de 2020.

[9] *React* Native disponible en: https://reactnative.dev/. Accedido en febrero de 2020.

[10] *Expo* disponible en: https://expo.io/. Accedido en febrero de 2020.

[11] *Yarn* disponible en: https://yarnpkg.com/. Accedido en febrero de 2020.

[12] *Node* disponible en: https://nodejs.org/es/. Accedido en febrero de 2020.

[13] *Docker* disponible en: https://www.docker.com/. Accedido en julio de 2020.

[14] *Git* disponible en: https://git-scm.com/. Accedido en febrero de 2020.

[15] *Firebase* disponible en: https://firebase.google.com/. Accedido en julio de 2020.

[16] *React Native Firebase* disponible en: https://rnfirebase.io/. Accedido en julio de 2020.

[17] NFC disponible en: https://www.xataka.com/moviles/nfc-que-es-y-para-que-sirve. Accedido en febrero de 2020.

[18] TPV disponible en: https://www.universalpay.es/que-es-tpv-como-funciona/. Accedido en febrero de 2020.

[19] TPV Virtual disponible en: https://www.clavei.es/blog/tpv-virtual-que-es-y-para-que-sirve/. Accedido en febrero de 2020.

[20] *Bizum* disponible en: https://bizum.es/. Accedido en febrero de 2020.

[21] *GitHub* disponible en: https://github.com/. Accedido en febrero de 2020.

[22] Scrum disponible en: https://proyectosagiles.org/que-es-scrum/. Accedido en febrero de 2020.

[23] Scrum disponible en: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software) . Accedido en febrero de 2020.

[24] *Contactless* disponible en: https://www.pibank.es/contactless-que-es/. Accedido en febrero de 2020.

[25] *GenyMotion* disponible en: https://www.genymotion.com/. Accedido en junio de 2020.

[26] *React Navigation* disponible en: https://reactnavigation.org/. Accedido en junio de 2020.

[27] react-native-maps disponible en: https://github.com/react-native-community/react-native-maps. Accedido en junio de 2020.

[28] tipsi-stripe disponible en: https://tipsi.github.io/tipsi-stripe/docs/index.html. Accedido en agosto de 2020.

[29] react-native-payments disponible en: https://github.com/naoufal/react-native-payments. Accedido en agosto de 2020.

[30] VisualStudio Code disponible en: https://code.visualstudio.com/. Accedido en febrero de 2020.

[31] *JavaScript* disponible en: https://www.javascript.com/. Accedido en febrero de 2020.

[32] Edutin Academy disponible en: https://edutin.com/. Accedido en febrero de 2020.

[33] Información buscada en: https://docs.expo.io/workflow/expo-cli/. Accedido en julio de 2020.

[34] *Stripe* disponible en: https://stripe.com/es. Accedido en julio de 2020.